COMO ESCRIBIR JUEGOS PARA EL ZX-SPECTURM

http://old8bits.blogspot.com/2016/04/como-escribir-juegos-para-el-zx_12.html

Capítulo 0: Introducción

Introducción

Ya has leído la documentación sobre el Z80, ya sabes cómo las operaciones afectan a los registros, y ahora quieres poner este conocimiento en práctica. A juzgar por el número de correos electrónicos que he recibido preguntando cómo leer el teclado, como calcular direcciones de la pantalla o como se emite ruido blanco por el altavoz, es evidente que en realidad no hay mucha información sobre recursos para el nuevo programador de Spectrum. Este documento, espero, crecerá para llenar este vacío en el momento oportuno. En su estado actual está claramente lejos de su finalización, pero con la publicación de los capítulos básicos que existen hasta la fecha espero que sea de ayuda para otros programadores.

El ZX Spectrum fue lanzado en abril de 1982, y para los estándares de hoy en día es una máquina primitiva. En el Reino Unido y otros países fue la máquina de juegos más popular de la década de 1980, y usando emuladores muchas personas están disfrutando de un viaje nostálgico en el tiempo a los juegos de su infancia. Otros ahora están descubriendo la máquina por primera vez, y algunos incluso están asumiendo el reto de escribir juegos para este pequeño y sencillo ordenador. Después de todo, si puedes escribir un juego decente en código de máquina para un equipo de 1980, probablemente no hay ninguno para el que no lo puedas escribir.

Los puristas odian este documento, pero escribir un juego no se trata de escribir "perfecto" código máquina del Z80 - como si existiera tal cosa. Un juego de Spectrum es una empresa importante, y no lo vas a terminar si estás demasiado obsesionado con la escritura de los mejores algoritmos para los marcadores o para la lectura del teclado. Una vez que has escrito una rutina que funciona y que no causa problemas en otros lugares, pasar a la siguiente rutina. No importa si está un poco desordenada o es ineficiente, ya que la parte importante es conseguir que funcione el juego. Nadie en su sano juicio va a desmontar el código y encontrarte los fallos.

Los capítulos de este documento se han ordenado para que el lector pueda empezar a escribir un simple juego tan pronto como sea posible. Nada es mejor que la emoción de la escritura de tu primer juego completo en código máquina, y he establecido este manual de tal manera que se cubran los requisitos mínimos muy básicos para esto en los primeros capítulos. A partir de ahí se pasa a métodos más avanzados que han de permitir al lector mejorar la calidad de los juegos que es capaz de escribir.

A lo largo de este documento se ha realizado una serie de supuestos. Para empezar, se supone que el lector está familiarizado con la mayoría de los códigos de operación del Z80 y lo que hacen. Si no es así, hay un montón de guías que explican esto mucho mejor de lo que yo lo podía hacer. Aprender instrucciones de código máquina no es difícil, pero saber cómo ponerlos juntos de manera significativa puede serlo. Familiarizarse con las instrucciones de carga **(ld),** comparación **(cp),** y salto condicional **(jp z / jp c / jp nc)** es un buen sitio para comenzar. El resto irán a su lugar una vez que estas se aprenden.

Herramientas

Hoy en día tienes el beneficio del hardware más sofisticado, ya no hay necesidad de desarrollar software en la máquina para la que está destinado. Hay un montón de ensambladores cruzados adecuados que te permitirán desarrollar el software del Spectrum en un PC y el archivo binario producido puede ser importado en un emulador - SPIN es un emulador popular que tiene soporte para esta función.

Para los gráficos hay una herramienta que yo uso llamada SevenUp, y lo recomiendo encarecidamente. Puede convertir mapas de bits en imágenes del Spectrum, y permite al programador especificar el orden en que se clasifican los sprites u otros gráficos. La salida puede ser en forma de una imagen binaria, o con código fuente en ensamblador. Otro programa popular es TommyGun.

Para la música recomiendo SoundTracker, utilidad que se puede descargar de los archivos de World of Spectrum. También necesitará un programa compilador por separado. Hay que tener en cuenta que estos son programas de Spectrum, no herramientas de PC y necesitan ser ejecutados en un emulador. Beepola es una excelente herramienta para producir música por el altavós de los 48K, y funciona en un PC con Windows.

Como editores y compiladores cruzados no estoy en condiciones de recomendar el mejor disponible, ya que uso un editor arcaico y un macro ensamblador cruzado para Z80 escrito en 1985, que se ejecuta en una ventana MS.DOS. Ninguno de los dos son herramientas que recomendaría a otros. Si necesita asesoramiento sobre herramientas que podrían ser adecuados para ti, te sugiero que acudas a los foros de desarrollo de World of Spectrum. Esta comunidad amigable tiene una amplia gama de niveles de experiencia y siempre está dispuesta a ayudar (**NDT**: en castellano puedes pasarte por Va de Retro y te ayudaremos).

Peculiaridades personales

Durante los muchos años que he estado escribiendo software para Spectrum he adquirido una serie de hábitos que pueden parecer extraños. La forma en que ordeno mis coordenadas, por ejemplo, no sigue las convenciones de las matemáticas. Mis programas en código máquina siguen la convención del BASIC de Sinclair **PRINT AT x,y**; donde x se refiere al número de celdas de caracteres o píxeles desde la parte superior de la pantalla, e y es el número de caracteres o píxeles desde el borde izquierdo. Si esto parece complicado al principio me disculpo, pero siempre parecía una manera más lógica de ordenar las cosas que se me quedó grabada. Algo de mi metodología puede parecer inusual en otros lugares, así que donde se puedes concebir una manera mejor de hacer algo en su lugar.

Otra cosa importante, comentar tu código a medida que avanzas es importante, si no esencial. Puede ser infernalmente difícil tratar de encontrar un error en una rutina sin comentar que escribiste hace tan sólo unas semanas. Puede parecer tedioso tener que documentar cada subrutina que escribes, pero ahorrarás tiempo de desarrollo a largo plazo. Además, si deseas volver a utilizar una rutina en otro juego en algún momento en el futuro, será muy fácil extraer la sección requerida y adaptarla para tu próximo proyecto.

Aparte de eso, simplemente diviértete. Si tienes alguna sugerencia o informar de errores, por favor ponte en contacto.

Capítulo 1: Texto y gráficos simples

Hola Mundo

El primer programa BASIC que los programadores novatos escriben por lo general son estas líneas:

```
10 PRINT "Hola Mundo"
20 GOTO 10
```

Muy bien, el texto puede ser diferente. Tu primer esfuerzo pudo ser "David es el mejor" o "Roberto estuvo aquí", pero seamos sinceros, la visualización de texto y gráficos en pantalla es probablemente el aspecto más importante de escribir cualquier juego de ordenador y (con la excepción de las máquinas de pinball o las tragaperras) es prácticamente imposible concebir un juego sin pantalla. Con esto en mente, vamos a empezar este tutorial con algunas de las rutinas de pantalla más importantes de la ROM del Spectrum.

Entonces, ¿cómo hacemos para convertir el programa BASIC anterior a código máquina? Bueno, podemos imprimir mediante el uso de la instrucción **RST 16,** lo que efectivamente es lo mismo que **PRINT CHR$ a** solo que simplemente imprime el carácter que ocupa el acumulador en el canal actual. Para imprimir una cadena en la pantalla, tenemos que llamar a dos rutinas: una para abrir la pantalla superior para la impresión (canal 2), y una segunda para imprimir la cadena. La rutina en la dirección ROM 5633 abrirá el número de canal que pasamos en el acumulador, y la de 8252 imprimirá una cadena que comienza en **de** con la longitud indicada en **bc** por este canal. Una vez que se abre el canal 2, toda la impresión se envía a la pantalla superior hasta que llamamos a 5633 con otro valor para enviar la salida a otra parte. Otros canales interesantes son 1 para la pantalla inferior (como **PRINT #1** en BASIC, que podemos usar para mostrar en las dos líneas inferiores) y el 3 para la ZX Printer.

```
       ld a,2             ; pantalla superior.
       call 5633          ; abrir canal.
loop   ld de,string       ; dirección de la cadena de caracteres.
       ld bc,eostr-string ; longitud de la cadena a imprimir.
       call 8252          ; imprimir su cadena.
       jp loop            ; repetir hsta que se llene la pantalla.

string defb '(tu nombre) es sexi'
eostr  equ $
```

Descarga

La ejecución de este programa llena la pantalla con el texto hasta que aparece **scroll?** en la parte inferior. Notarás sin embargo que, en lugar de que cada línea de texto aparezca en una línea propia como en el listado, el principio de cada cadena continúa directamente desde el final de la anterior, que no es exactamente lo que queríamos. Para lograr esto tenemos que lanzar un salto de línea

nosotros mismos, utilizando un código de control ASCII. Una forma de hacer esto sería cargar el acumulador con el código para una nueva línea (13), a continuación utilizar **RST 16** para imprimir ese código. Otra forma más eficiente es agregar el código ASCII al final de nuestra cadena de este modo:

```
string DEFB '(tu nombre) es sexi'
       DEFB 13
eostr  equ $
```

Descarga

Hay una serie de códigos de control ASCII como éste que altera la posición de impresión actual, los colores, etc., y la experimentación te ayudará a decidir cuáles encontrarás más útiles. Estos son los principales que utilizo:

```
13      (NEWLINE) Establece la posición de impresión al principio de la línea

                  siguiente.
16,c    (INK)     Establece el color de la tinta con el valor del siguiente
byte.
17,c    (PAPER)   Ajusta el color del fondo con el valor del siguiente byte.
22,x,y  (AT)      Ajusta la impresión a las coordenadas x e y con los valores

                  especificados en los dos bytes siguientes.
```

El código 22 es particularmente útil para establecer las coordenadas en las que se va a mostrar un texto o un carácter gráfico. En este ejemplo se mostrará un signo de exclamación en la parte inferior derecha de la pantalla:

```
       ld a,2              ; pantalla superior.
       call 5633           ; abrir canal.
       ld de,string        ; dirección de la cadena.
       ld bc,eostr-string  ; longitud de la cadena a imprimir.
       call 8252           ; imprimir nuestra cadena.
       ret

string defb 22,21,31,'!'
eostr  equ $
```

Descarga

Este programa va un paso más allá y anima un asterisco desde el fondo hasta la parte superior de la pantalla:

```
       ld a,2          ; 2 = pantalla superior.
       call 5633       ; abrir el canal.
       ld a,21         ; fila 21 = parte inferior de la pantalla.
```

```
       ld (xcoord),a   ; establece la coordenada x inicial.
loop   call setxy      ; actualiza nuestras coordenadas x/y.
       ld a,'*'        ; queremos un asterisco aquí.
       rst 16          ; presentarlo.
       call delay      ; esperamos un retardo.
       call setxy      ; actualiza nuestras coordenadas x/y.
       ld a,32         ; código ASCII para espacio en blanco.
       rst 16          ; borrar el anterior asterisco.
       ld hl,xcoord    ; posición vertical.
       dec (hl)        ; moverla hacia arriba una línea.
       ld a,(hl)       ; ¿donde estamos ahora?
       cp 255          ; ¿nos pasamos del inicio de la pantalla?
       jr nz,loop      ; no, repetir.
       ret
delay  ld b,10         ; longitud del retardo.
delay0 halt            ; esperar una interrupción.
       djnz delay0     ; bucle.
       ret             ; regresar.
setxy  ld a,22         ; código de control ASCII para AT.
       rst 16          ; imprimir aquí.
       ld a,(xcoord)   ; posición vertical.
       rst 16          ; imprimir aquí.
       ld a,(ycoord)   ; coordenada y.
       rst 16          ; imprimir aquí.
       ret

xcoord defb 0
ycoord defb 15
```

Descarga

Impresión de gráficos simples

Mover asteriscos alrededor de la pantalla está muy bien, pero incluso para el juego más simple lo que realmente necesitamos es mostrar gráficos. Se hablará de gráficos avanzados en capítulos posteriores, por ahora sólo vamos a usar simples invasores del espacio como tipo de gráficos, y como cualquier programador en BASIC te dirá, el Spectrum tiene un mecanismo muy simple para esto: los gráficos definidos por el usuario, generalmente abreviados como UDG.

La tabla ASCII del Spectrum contiene 21 (19 en modo 128k) caracteres gráficos definidos por el usuario, comenzando en el código 144 y llegando hasta el 164 (162 en modo 128k). En BASIC los UDG se definen introduciendo datos mediante **POKE** en el área de UDG de la parte superior de la RAM, pero en código de máquina tiene más sentido cambiar la variable de sistema que apunta a la posición de memoria en la que se almacenan los UDG, lo que se realiza cambiando el valor de dos bytes en la dirección 23675. Ahora podemos modificar nuestro programa del asterisco en movimiento para mostrar en su lugar un gráfico con los cambios que están subrayados.

```
       ld hl,udgs      ; UDG que definimos.
       ld (23675),hl   ; actualizamos la variable del sistema para los UDG.
       ld a,2          ; 2 = pantalla superior.
       call 5633       ; abrir el canal.
       ld a,21         ; fila 21 = parte inferior de la pantalla.
```

```
       ld (xcoord),a    ; establece la coordenada x inicial.
loop   call setxy       ; actualiza nuestras coordenadas x/y.
       ld a,144         ; en lugar del asterisco aquí usamos un UDG.
       rst 16           ; presentarlo.
       call delay       ; esperamos un retardo.
       call setxy       ; actualiza nuestras coordenadas x/y.
       ld a,32          ; código ASCII para espacio en blanco.
       rst 16           ; borrar el anterior asterisco.
       ld hl,xcoord     ; posición vertical.
       dec (hl)         ; moverla hacia arriba una línea.
       ld a,(hl)        ; ¿donde estamos ahora?
       cp 255           ; ¿nos pasamos del inicio de la pantalla?
       jr nz,loop       ; no, repetir.
       ret
delay  ld b,10          ; longitud del retardo.
delay0 halt             ; esperar una interrupción.
       djnz delay0      ; bucle.
       ret              ; regresar.
setxy  ld a,22          ; código de control ASCII para AT.
       rst 16           ; imprimir aquí.
       ld a,(xcoord)    ; posición vertical.
       rst 16           ; imprimir aquí.
       ld a,(ycoord)    ; coordenada y.
       rst 16           ; imprimir aquí.
       ret

xcoord defb 0
ycoord defb 15
udgs   defb 60,126,219,153
       defb 255,255,219,219
```

Descarga

Por supuesto, no hay razón por la que no se puedan utilizar más de 21 UDG si se desea. Sólo hay que establecer una serie de bancos de ellos en memoria y apuntar a cada uno cuando se necesite.

Como alternativa, se puede redefinir el conjunto de caracteres en su lugar. Esto nos da un mayor rango de caracteres ASCII desde 32 (ESPACIO) a 127 (el símbolo de copyright). Incluso se puede mezclar texto y gráficos, se redefinen las letras y los números con una fuente del estilo de tu elección, y a continuación utilizamos los símbolos y letras minúsculas para los aliens, zombies o lo que el juego requiera. Para apuntar a otro conjunto restamos 256 de la dirección en la que se coloca la fuente y colocamos esto en la variable de sistema de dos bytes en la dirección 23606. El tipo de letra predeterminado de Sinclair, por ejemplo, se encuentra en la dirección ROM 15616, por lo que la variable de sistema en la dirección 23606 apunta a 15360 cuando el Spectrum se enciende.

Este código copia la fuente Sinclair de la ROM a la RAM y lo "pasa a negrita" a continuación, luego establece la variable de sistema para apuntar al nuevo juego de caracteres:

```
       ld hl,15616          ; fuente en ROM.
       ld de,60000          ; dirección de su fuente.
       ld bc,768            ; 96 caracteres * 8 filas que alterar.
font1  ld a,(hl)            ; obtiene el bitmap.
       rlca                 ; lo rota hacia la izquierda.
```

```
       or (hl)              ; combina las 2 imágenes.
       ld (de),a            ; lo escribe en la nueva fuente.
       inc hl               ; siguiente byte del antiguo.
       inc de               ; siguiente byte del nuevo.
       dec bc               ; decrementar el contador.
       ld a,b               ; byte alto.
       or c                 ; combinar con el byte bajo.
       jr nz,font1          ; repetir hasta que bc=zero.
       ld hl,60000-256      ; fuente menos 32*8.
       ld (23606),hl        ; apuntar a la nueva fuente.
       ret
```

Descarga

Mostrando números

Para la mayoría de los juegos lo mejor para definir la puntuación del jugador es como una cadena de dígitos ASCII, aunque eso significa más trabajo en las rutinas de puntuación y hace que las tablas de puntuación más alta sean un verdadero grano en el culo para un programador en lenguaje ensamblador sin experiencia. Vamos a cubrir esto en un capítulo posterior, pero por ahora vamos a utilizar algunas rutinas de la ROM que están muy a mano para imprimir nuestros números.

Hay dos formas de imprimir un número en la pantalla, la primera es hacer uso de la misma rutina que la ROM utiliza para imprimir los números de línea del Sinclair BASIC. Para esto basta con cargar el par de registros **bc** con el número que desea imprimir, y luego llamar a 6683:

```
       ld bc,(puntuación)
       call 6683
```

Sin embargo, dado que los números de línea en el BASIC sólo pueden llegar al 9999, tiene la desventaja de que sólo será capaz de mostrar un número de cuatro dígitos. Una vez que la puntuación del jugador llega a 10000 se muestran caracteres ASCII en lugar de números. Afortunadamente, hay otro método que llega mucho más allá. En lugar de llamar a la rutina de visualización del número de línea, podemos llamar a una rutina para colocar el contenido de los registros **bc** en la pila de cálculo, y a continuación llamar a otra rutina que muestra el número en la parte superior de esta pila. No te preocupes de lo que es la pila de cálculo y cuál es su función, porque es de poca utilidad para un programador de juegos arcade, pero ya que podemos hacer uso de ella lo haremos. Sólo recuerda que las tres líneas siguientes mostrará un número entero del 0 al 65535 ambos inclusive:

```
       ld bc,(puntuación)
       call 11563           ; apilar número en bc.
       call 11747           ; mostrar la cima de la pila de cálculo.
```

Descarga los dos trozos de código

Cambio de Colores

Para establecer la tinta, el fondo, el brillo y el flash a niveles permanentes podemos escribir directamente en la variable de sistema en 23693, a continuación borra la pantalla con una llamada a la ROM:

```
; Queremos una pantalla de color amarillo.

       ld a,49             ; tinta azul (1) con fondo amarillo (6 * 8).
       ld (23693),a        ; establecer nuestros colores de pantalla.
       call 3503           ; borrar la pantalla.
```

La forma más rápida y sencilla de establecer el color del borde es escribir en el puerto 254. Los 3 bits menos significativos del byte que enviamos determinan el color, por lo que para establecer el borde en rojo:

```
       ld a,2              ; 2 es el código para el rojo.
       out (254),a         ; escribir en el puerto 254.
```

El puerto 254 también maneja el altavoz y el micrófono a través de los bits 3 y 4, por tanto el efecto del borde sólo durará hasta la próxima llamada a la rutina de sonido del altavoz en la ROM (más adelante hablaremos de esto), por lo que es necesaria una solución más permanente. Para ello, simplemente hay que cargar el acumulador con el color requerido y llamar a la rutina de la ROM en 8859. Esto cambiará el color y la variable de sistema BORDCR (ubicada en la dirección 23624) en consonancia. Para establecer un borde rojo permanente podemos hacer esto:

```
       ld a,2              ; 2 es el código para el rojo.
       call 8859           ; establecer el color del borde.
```

Descarga los tres trozos de código

Capítulo 2: Control del teclado y el Joystick

Una tecla a la vez

A condición de que no hayas deshabilitado las interrupciones o te hayas entrometido con ellas, por defecto la ROM del Spectrum leerá automáticamente el teclado y actualizará varias variables del sistema situadas en la posición de memoria 23552 cincuenta veces por segundo. La forma más sencilla de comprobar si hay una pulsación de tecla es primero cargar la dirección 23560 con un valor nulo, y luego interrogar esta ubicación hasta que cambie, el resultado es el valor ASCII de la tecla pulsada. Esto es muy útil para aquellas situaciones de "presione cualquier tecla para continuar", para la elección de elementos de un menú, o para la entrada del nombre en las rutinas de puntuación más alta. Una rutina de este tipo podría tener este aspecto:

```
       ld hl,23560         ; variable del sistema LAST K.
       ld (hl),0           ; poner un valor nulo aquí.
loop   ld a,(hl)           ; nuevo valor de LAST K.
       cp 0                ; ¿sique a cero?
       jr z,loop           ; si, no se ha pulsado nada.
       ret                 ; tecla pulsada.
```

Descargar código con un ejemplo

Pulsación de varias teclas

Las pulsaciones de teclas individuales son raras de usar para la rápida acción de los juegos arcade, en su lugar necesitamos detectar más de una pulsación simultánea, y aquí es donde las cosas se ponen un poco más complicadas. En lugar de leer direcciones de memoria tenemos que leer uno de ocho puertos, cada uno de los cuales corresponde a una fila de cinco teclas. Por supuesto, la mayoría de modelos de Spectrum parecen tener muchas mas teclas ¿como se conectan? Bueno, en realidad no lo hacen. La distribución del teclado del Spectrum original consistía en sólo cuarenta teclas, dispuestas en ocho grupos de filas de cinco teclas. Con el fin de acceder a algunas de las funciones era necesario presionar ciertas combinaciones de teclas al mismo tiempo, por ejemplo para borrar la combinación requerida eran **CAPS SHIFT** y **0** a la vez. Sinclair añadió estas teclas extra cuando llegó el Spectrum Plus a escena en 1985, y funcionan mediante la simulación de la pulsación simultánea de las combinaciones de pulsaciones de teclas necesarias en los modelos de teclado de goma original.

La distribución del teclado original se separa en estas agrupaciones:

Puerto	Teclas
32766	B, N, M, Symbol Shift, Space
49150	H, J, K, L, Enter
57342	Y, U, I, O, P
61438	6, 7, 8, 9, 0

```
63486   5, 4, 3, 2, 1
64510   T, R, E, W, Q
65022   G, F, D, S, A
65278   V, C, X, Z, Caps Shift
```

Para descubrir qué teclas están siendo presionadas leemos el número del puerto adecuado, cada tecla en la fila está asignanda a uno de los cinco bits bajos **D0-D4** (valores 1, 2, 4, 8 y 16) donde **D0** representa la tecla exterior y **D4** la más interna. Curiosamente, cada bit está en alto (1) cuando no se presiona, y en bajo (0) cuando lo está, al contrario de lo que cabría esperar.

Para leer una fila de cinco teclas simplemente cargamos el número de puerto en el par de registros **BC**, a continuación realizar la instrucción **in a,(c)**. Como sólo queremos los bits de menor valor podemos ignorar los bits que no deseamos, ya sea con una **and 31** o mediante la rotación de los bits de salida del acumulador usando la bandera de acarreo mediante cinco instrucciones **rra: call c, (dirección)**.

Si esto es difícil de entender revisa el siguiente ejemplo:

```
       ld bc,63486          ; fila del teclado 1-5/puerto de joystick 2.
       in a,(c)             ; ver que tecla se ha pulsado.
       rra                  ; bit externo = key 1.
       push af              ; recordar el valor.
       call nc,mpl          ; si se ha pulsado, moverse a la izquierda.
       pop af               ; restaurar el acumulador.
       rra                  ; siguiente bit (valor 2) = key 2.
       push af              ; recordar el valor.
       call nc,mpr          ; se ha pulsado, moverse a la derecha.
       pop af               ; restaurar el acumulador.
       rra                  ; siguiente bit (valor 4) = key 3.
       push af              ; recordar el valor.
       call nc,mpd          ; se ha pulsado, moverse hacia abajo.
       pop af               ; restaurar el acumulador.
       rra                  ; siguiente bit (valor 8) reads key 4.
       call nc,mpu          ; se ha pulsado, moverse hacia arriba.
```

Ver mas abajo apartado "un juego simple" para un ejemplo de esto

Joysticks

Los puertos de joystick Sinclair 1 y 2 fueron simplemente asignados a cada una de las filas de las teclas numéricas, lo que se puede ver fácilmente desde el editor de BASIC, usando el joystick para escribir números. El puerto 1 (interface 2) se asigna a las teclas 6, 7, 8, 9 y 0, el puerto 2 (interface 1) a las teclas 1, 2, 3, 4 y 5. Para detectar la entrada de joystick leemos el puerto del mismo modo que para la lectura del teclado. Los joysticks Sinclair utilizan los puertos 63486 (interface 1 / puerto 2) y 61438 (interface 2 / puerto 1), los bits **D0-D4** darán un 0 para el presionado, 1 para no presionado.

El popular formato de joystick Kempston no está asignada al teclado y se puede leer en su lugar mediante el puerto 31. Esto significa que podemos usar un sencillo **in a,(31)**. Una vez más, se utilizan los valores de los bits D0-D4, aunque esta vez los valores son como se podría esperar, con un valor alto si se está aplicando el joystick en una dirección particular. Los valores de los bits resultantes serán 1 para presionado, 0 para no presionado.

```
; Ejemplo de rutina de control del joystick.

joycon ld bc,31            ; puerto del joystick Kempston.
       in a,(c)            ; leer la entrada.
       and 2               ; verificar el bit "izquierda".
       call nz,joyl        ; moverse a la izquierda.
       in a,(c)            ; leer la entrada.
       and 1               ; verificar el bit "derecha".
       call nz,joyr        ; moverse a la derecha.
       in a,(c)            ; leer la entrada.
       and 8               ; verificar el bit "arriba".
       call nz,joyu        ; moverse arriba.
       in a,(c)            ; leer la entrada.
       and 4               ; verificar el bit "abajo".
       call nz,joyd        ; moverse abajo.
       in a,(c)            ; leer la entrada.
       and 16              ; verificar el bit de disparo.
       call nz,fire        ; disparo pulsado.
```

Un juego simple

Ahora podemos ir un paso más allá y, poniendo en práctica lo que ya hemos cubierto, escribir la sección de control principal para un juego básico. Esta será la base de una simple variante del Centipede (ciempiés) que vamos a desarrollar en los próximos capítulos. No hemos cubierto todo lo necesario para un juego todavía, pero podemos hacer un comienzo con un pequeño bucle de control que permite al jugador manipular una pequeña base de disparo por la pantalla. Te lo advierto, este programa no tiene salida al BASIC así que asegúrate de haber guardado una copia del código fuente antes de ejecutarlo.

```
; Queremos una pantalla en negro.

       ld a,71             ; tinta blanca (7) en fondo negro (0),
                           ; con brillo (64).
       ld (23693),a        ; establecer nuestros colores de pantalla.
       xor a               ; forma rápida de cargar el acumulador con cero.
       call 8859           ; establecer el colore del borde permanente.

; Configurar los gráficos.

       ld hl,blocks        ; dirección de los datos para los UDG.
       ld (23675),hl       ; apuntar los UDG hacia aquí.

; De acuerdo, vamos a empezar el juego.

       call 3503           ; rutina ROM - borra la pantalla, abre el canal 2.

; Inicializar coordenadas.
```

```
       ld hl,21+15*256      ; cargar el par hl con las coordenadas iniciales.
       ld (plx),hl          ; fijar las coordenadas del jugador.

       call basexy          ; establecer las posiciones x e y del jugador.
       call splayr          ; mostrar símbolo de la base del jugador.

; Este es el bucle principal.

mloop  equ $

; Borrar el jugados.

       call basexy          ; establecer las posiciones x e y del jugador.
       call wspace          ; mostrar un espacio sobre el jugador.

; Ahora hemos eliminado el jugador y lo movemos antes de volverlo a mostrar
; en sus nuevas coordenadas.

       ld bc,63486          ; fila del teclado 1-5/joystick puerto 2.
       in a,(c)             ; ver que teclas están pulsadas.
       rra                  ; bit mas externo = tecla 1.
       push af              ; recordar el valor.
       call nc,mpl          ; si está está siendo pulsada, moverse a la
izquierda.
       pop af               ; restaurar el acumulador.
       rra                  ; siguiente bit (valor 2) = tecla 2.
       push af              ; recordar el valor.
       call nc,mpr          ; si está está siendo pulsada, moverse a la derecha.
       pop af               ; restaurar el acumulador.
       rra                  ; siguiente bit (valor 4) = tecla 3.
       push af              ; recordar el valor.
       call nc,mpd          ; si está está siendo pulsada, moverse hacia abajo.
       pop af               ; restaurar el acumulador.
       rra                  ; siguiente bit (valor 8) = tecla 4.
       call nc,mpu          ; si está está siendo pulsada, moverse hacia arriba.

; Ahora que se ha movido podemos volver a mostrar al jugador.

       call basexy          ; establecer las posiciones x e y del jugador.
       call splayr          ; mostrar al jugador.

       halt                 ; retardo.

; Saltar de nuevo al principio del bucle principal.

       jp mloop

; Mover al jugador a la izquierda.

mpl    ld hl,ply            ; recordar, ¡y es la coordenada horizontal!
       ld a,(hl)            ; ¿cuál es el valor actual?
       and a                ; ¿es cero?
       ret z                ; sí - no podemos seguir hacia la izquierda.
       dec (hl)             ; restar 1 a la coordenada y.
       ret

; Mover al jugador a la derecha.

mpr    ld hl,ply            ; recordar, ¡y es la coordenada horizontal!
       ld a,(hl)            ; ¿cuál es el valor actual?
```

```
       cp 31              ; ¿está en el borde derecho (31)?
       ret z              ; sí - no podemos seguir hacia la derecha.
       inc (hl)           ; sumar 1 a la coordenada y.
       ret

; Mover al jugador hacia arriba.

mpu    ld hl,plx          ; recordar, ¡x es la coordenada vertical!
       ld a,(hl)          ; ¿cuál es el valor actual?
       cp 4               ; ¿está en el límite superior (4)?
       ret z              ; sí - no podemos seguir hacia arriba.
       dec (hl)           ; restar 1 a la coordenada x.
       ret

; Mover al jugador hacia abajo.

mpd    ld hl,plx          ; recordar, ¡x es la coordenada vertical!
       ld a,(hl)          ; ¿cuál es el valor actual?
       cp 21              ; ¿está en el límite inferior (21)?
       ret z              ; sí - no podemos seguir hacia abajo.
       inc (hl)           ; sumar 1 a la coordenada x.
       ret

; Configurar las coordenadas X e Y de la posición de la base del jugador,
; se le llama antes de la visualización y supresión de la base.

basexy ld a,22            ; código para AT.
       rst 16
       ld a,(plx)         ; coordenada vertical del jugador.
       rst 16             ; fijar la posición vertical del jugador.
       ld a,(ply)         ; coordenada horizontal del jugador.
       rst 16             ; fijar la posición horizontal del jugador.
       ret

; Mostrar al jugador en la posición de impresión actual.

splayr ld a,69            ; tinta cían (5) en fondo negro (0),
                          ; brillante (64).
       ld (23695),a       ; establecer nuestros colores temporales de pantalla.
       ld a,144           ; código ASCII para el UDG 'A'.
       rst 16             ; dibujar jugador.
       ret

wspace ld a,71            ; tinta blanca (7) en fondo negro (0),
                          ; brillante (64).
       ld (23695),a       ; establecer nuestros colores temporales de pantalla.
       ld a,32            ; carácter de ESPACIO.
       rst 16             ; dibujar espacio.
       ret

plx    defb 0             ; coordenada x del jugador.
ply    defb 0             ; coordenada y del jugador.

; gráficos UDG.

blocks defb 16,16,56,56,124,124,254,254    ; base del jugador.
```

Descarga el código

Rápido, ¿no es así? De hecho, hemos reducido el ritmo con una instrucción de interrupción, pero todavía funciona a unos veloces 50 cuadros por segundo, lo que es probablemente un poco demasiado rápido. No te preocupes, ya que seguiremos añadiendo más características al código que comenzarán a reducir la velocidad. Si te sientes cómodo puedes intentar adaptar el programa anterior para trabajar con un joystick Kempston. No es difícil, requiere simplemente cambiar el puerto 63486 por el puerto 31, y sustituir los cuatro posteriores **call nc,(dirección)** por **call c,(dirección)** (¿recuerdas que los bits se invierten?)

Usar teclas redefinibles es un poco más complicado. Como probablemente sabes, el teclado del Spectrum original se divide en 8 filas de 5 teclas cada uno, leyendo el puerto asociado con una determinada fila de teclas, y a continuación probando los bits **D0-D4** podemos saber si se pulsa una tecla en particular. Si vas a sustituir **ld bc,31** en el fragmento de código anterior con **ld bc,49150** podría acceder a saber si la fila de teclas se ha usado, aunque esto no lo hace una buena rutina de teclas redefinibles. Afortunadamente, hay otra manera de hacer las cosas.

Podemos establecer el puerto necesario para cada fila de teclas utilizando la fórmula del manual del Spectrum . Donde n es el número de fila 0-7, la dirección del puerto será $254 + 256 * (255 - 2^n)$. Hay una rutina ROM en la dirección 654 que hace un montón de trabajo duro para nosotros, devolviendo el número de la tecla pulsada en el registro **e**, en el rango de 0-39. Un valor 0-7 corresponde a la tecla más interna de cada fila (eso es B, H, Y, 6, 5, T, G y V), un valor 8-15 a la siguiente tecla a lo largo de cada fila, y así hasta el 39 para la tecla externa en la última fila (**CAPS SHIFT**). El estado de la tecla de maúsculas a su vez también se devuelve en el registro **d**. Si no se pulsa ninguna tecla retorna en **e** el valor 255.

La rutina de la ROM sólo puede devolver un único número de tecla, lo que no es bueno para la detección de más de una tecla a la vez. Para determinar si una tecla específica está siendo presionada o no en cualquier momento, es necesitamos convertir el número de nuevo en un puerto y, a continuación, leer ese puerto y comprobar el bit individual por nosotros mismos. Hay una rutina muy útil que utilizo para mis trabajo, y es la única rutina en mis juegos que no he escrito yo mismo. El crédito por eso debe ir a **Stephen Jones**, un programador que escribió excelentes artículos para el ***Spectrum Club Discovery*** hace muchos años. Para utilizar su rutina, cargar el acumulador con el número de la tecla que deseas probar, llama a **ktest**, a continuación, comprueba la bandera de acarreo. Si se ha establecido (1) no está siendo presionada la tecla, si no hay acarreo (0) es que se pulsa la tecla. Si eso es demasiado confuso y parece como una forma incorrecta de rodar, pon una instrucción **ccf** justo antes del **ret**.

```
; rutina de prueba del teclado del Sr. Jones.

ktest  ld c,a             ; tecla a probar en c.
       and 7              ; mascara de bits d0-d2 para la fila.
       inc a              ; en el rango de 1-8.
       ld b,a             ; colocarlo en b.
       srl c              ; dividir c por 8,
       srl c              ; encontrar la posición dentro de la fila.
       srl c
       ld a,5             ; sólo 5 teclas por fila.
```

```
       sub c               ; restar de la posición.
       ld c,a              ; ponerlo en c.
       ld a,254            ; byte alto del puerto a leer.
ktest0 rrca                ; rotar en su posición.
       djnz ktest0         ; repetir hasta que hemos encontrado la fila
correspondiente.
       in a,(254)          ; leer el puerto (a=alto, 254=bajo).
ktest1 rra                 ; rotar el bit fuera del resultado.
       dec c               ; bucle del contador.
       jp nz,ktest1        ; repetir hasta que el bit se posicione por acarreo.
       ret
```

Capítulo 3: Efectos de sonido por el altavoz

***NdT**: Podeis leer aquí mi artículo sobre el tema*

El altavoz

Hay dos formas de generar sonido y música en el ZX Spectrum, la mejor y más complicada es a través del chip de sonido AY 38912 en los modelos 128K. Este método se describe en detalle en un capítulo posterior, pero por ahora vamos a preocuparnos por el altavoz del 48K. Por simple que sea, este método tiene sus usos, especialmente para los efectos de sonido cortos y agudos durante los juegos.

Beep

En primer lugar tenemos que saber cómo producir un pitido de un determinado tono y duración, y la ROM de Sinclair tiene una rutina en la dirección 949 bastante accesible para hacer el trabajo por nosotros, todo lo que se requiere es pasar los parámetros de tono en el par de registro **HL** y la duración en **DE**, llamar a 949 y nos dará un "bip" apropiado.

Por desgracia, la forma de calcular los parámetros con los que trabaja es un poco complicada y requiere un poco de cálculo. Necesitamos conocer el valor en Hercios de la frecuencia de la nota a emitir, esencialmente es el número de veces que el altavoz tiene que activarse por segundo para producir el tono deseado. Una tabla adecuada sería ***(NdT**: El documento utiliza la codificación de letras, pongo también la nota*):

```
C central   261'63   DO medio
C sostenido 277'18   DO#
D           293'66   RE
D sostenido 311'13   RE#
E           329'63   MI
F           349'23   FA
F sostenido 369'99   FA#
G           392'00   SOL
G sostenido 415'30   SOL#
A           440'00   LA
A sostenido 466'16   LA#
B           493'88   SI
```

Para una octava más alta sólo tienes que duplicar la frecuencia, para una octava más baja reducirla a la mitad. Por ejemplo, para producir una nota C (DO) una octava más alta que el C central (DO medio) se toma el valor de C central (261'63) y se duplica a 523'26.

Una vez establecida la frecuencia, la multiplicamos por el número de segundos requeridos, y se le pasa a la rutina de la ROM en el par de registros **DE** como la duración. Por tanto para hacer sonar la nota C central (DO medio) durante una décima de segundo, la duración requerida sería 261'63 * 0'1 = 26. El tono es elaborado dividiendo 437500 por la frecuencia y restando 30'125, pasando el

resultado en los registros **HL**. Para un C central (DO medio) esto significaría un valor de 437500 / 261'63 - 30'125 = 1'642.

En otras palabras:

```
DE = Duración = Frecuencia * segundos
HL = Tono = 437500 / Frecuencia - 30'125
```

Así que para tocar un G sostenido (SOL#) una octava por encima de la del C central (DO medio) durante un cuarto de segundo:

```
; Frecuencia del G sostenido en la octava del C central = 415.30
; Frecuencia del G sostenido una octava más alta = 830.60
; Duración = 830.6 * 1/4 = 207.65
; Tono = 437500 / 830.6 - 30.125 = 496.6

       ld hl,497           ; tono.
       ld de,208           ; duración.
       call 949            ; rutina de beep de la ROM.
       ret
```

Por supuesto esta rutina no solo es útil para notas musicales, la podemos utilizar para una variedad de efectos, uno de mis favoritos es un rutina simple de inflexión de tono:

```
       ld hl,500           ; tono inicial
       ld b,250            ; longitud de la inflexión del tono.
loop   push bc
       push hl             ; guardar tono.
       ld de,1             ; duración muy corta.
       call 949            ; rutina de beep de la ROM.
       pop hl              ; restablecer el tono.
       inc hl              ; subir el tono.
       pop bc
       djnz loop           ; repetir.
       ret
```

Hacer sonar la rutina anterior es bastante fácil para ajustar el tono hacia arriba y abajo, y al cambiar frecuencia de inicio, inflexión de tono y duración se producen una serie de efectos interesantes. Una palabra de advertencia: no vaya demasiado a lo loco con sus valores de tono o la duración, o la rutina del beep se quedará bloqueada y no será capaz de recuperar el control de su Spectrum sin resetearlo.

Ruido blanco

Cuando se utiliza el altavoz ni siquiera hay que usar las rutinas de la ROM, es bastante fácil escribir nuestras propias rutinas de efectos de sonido, sobre todo si queremos generar ruido blanco para los choques y golpes. El ruido blanco es por lo general mucho más divertido para jugar.

Para generar ruido blanco todo lo que necesitamos es un generador de números aleatorios rápido y simple (una secuencia de Fibonacci podría funcionar, pero me gustaría recomendar usar un puntero sobre las primeras 8K de ROM e ir a buscar el byte en ese lugar para obtener una secuencia razonable de números al azar de 8 bit). A continuación, escribimos este valor en el puerto 254. Recuerda que este puerto también controla el color del borde, de modo que si no quieres un efecto de borde de rayas multicolor necesitamos enmascarar los bits del borde con **AND 248** y añadir el número del color de borde que queremos (1 para el azul, 2 para rojo, etc.) antes de realizar una instrucción **OUT (254)**. Cuando hemos hecho esto tenemos que poner un pequeño bucle de retardo (corto para tono alto, largo para tonos más bajos) y repetir el proceso unos pocos cientos de veces. Esto nos dará un buen efecto de "choque".

Esta rutina se basa en un efecto de sonido de Egghead 3:

```
noise  ld e,250            ; repetir 250 veces.
       ld hl,0             ; apunta al inicio de la ROM.
noise2 push de
       ld b,32             ; longitud del paso.
noise0 push bc
       ld a,(hl)           ; siguiente numero "aleatorio".
       inc hl              ; apuntar.
       and 248             ; queremos un borde blanco.
       out (254),a         ; salida al altavoz.
       ld a,e              ; como e disminuye...
       cpl                 ; ...incrementamos el retraso.
noise1 dec a               ; decrementar el contador del bucle.
       jr nz,noise1        ; bucle de retardo.
       pop bc
       djnz noise0         ; siguiente paso.
       pop de
       ld a,e
       sub 24              ; tamaño del paso.
       cp 30               ; fin del rango.
       ret z
       ret c
       ld e,a
       cpl
noise3 ld b,40             ; periodo de silencio.
noise4 djnz noise4
       dec a
       jr nz,noise3
       jr noise2
```

Capítulo 4: Números aleatorios

La generación de números aleatorios en código de máquina puede ser un problema difícil para un programador novato.

En primer lugar vamos a dejar una cosa clara, no hay una cosa que sea un generador de números aleatorios. La CPU se limita a seguir las instrucciones y no tiene mente propia, no puede simplemente sacar un número de la nada sobre la base de un capricho. En su lugar, tiene que seguir una fórmula que producirá una secuencia impredecible de números que no parecen seguir ningún tipo de patrón, y por lo tanto dan la impresión de aleatoriedad. Todo lo que podemos hacer es devolver un falso (o seudo) número aleatorio.

Un método de obtención de un número seudo-aleatorio sería el uso de la secuencia de Fibonacci, sin embargo el método más fácil y más rápido de generar un número de 8 bits seudo-aleatorio en el Spectrum es poner un puntero hacia la ROM, y examinar el contenidos del byte en ese lugar a su vez. Hay un pequeño inconveniente en este método, la ROM de Sinclair contiene una área muy uniforme y poco aleatoria hacia el final que es mejor evitar. Al limitar el puntero a, por ejemplo, los primeros 8K ROM tenemos una secuencia de 8192 números de "al azar", más que suficiente para la mayoría de los juegos. De hecho, todos los juegos que he escrito con un generador de números aleatorios utilizan este método, o uno muy similar:

```
; Sencillo generador de números seudo-aleatorio.
; Seguir un puntero a través de la ROM (a partir de una semilla),
; retornando el contenido del byte en esa posición.

random ld hl,(seed)        ; puntero
       ld a,h
       and 31              ; mantenerlo en los primeros 8Kb de ROM.
       ld h,a
       ld a,(hl)           ; Obtener el número "aleatorio" de esa ubicación.
       inc hl              ; Incrementar el puntero.
       ld (seed),hl
       ret
seed   defw 0
```

Vamos a poner nuestro nuevo generador de números aleatorios para utilizarlo en nuestro juego Centipede. Todos los juegos centipede necesita setas - muchas- dispersas al azar en todo el área del juego, y ahora podemos llamar a la rutina aleatoria para que nos suministre las coordenadas donde se muestra cada seta. Las partes resaltadas son los que tenemos que añadir.

```
; Queremos una pantalla en negro.

       ld a,71             ; tinta blanca (7) en fondo negro (0),
                           ; con brillo (64).
       ld (23693),a        ; establecer nuestros colores de pantalla.
       xor a               ; forma rápida de cargar el acumulador con cero.
       call 8859           ; establecer el colore del borde permanente.
```

```
; Configurar los gráficos.

       ld hl,blocks         ; dirección de los datos para los UDG.
       ld (23675),hl        ; apuntar los UDG hacia aquí.

; De acuerdo, vamos a empezar el juego.

       call 3503            ; rutina ROM - borra la pantalla, abre el canal 2.

; Inicializar coordenadas.

       ld hl,21+15*256      ; cargar el par hl con las coordenadas iniciales.
       ld (plx),hl          ; fijar las coordenadas del jugador.

       call basexy          ; establecer las posiciones x e y del jugador.
       call splayr          ; mostrar símbolo de la base del jugador.

; Ahora queremos llenar la zona de juegos con setas.

       ld a,68              ; tinta verde (4) en fondo negro (0),
                            ; con brillo (64).
       ld (23695),a         ; establecer nuestros colores temporales.
       ld b,50              ; comenzar con unas pocas.
mushlp ld a,22              ; código del carácter de control para AT.
       rst 16
       call random          ; obtener un número 'aleatorio'.
       and 15               ; en vertical en rango de 0 a 15.
       rst 16
       call random          ; obtener otro número seudo-aleatorio.
       and 31               ; horizontal en el rango de 0 a 31.
       rst 16
       ld a,145             ; el UDG 'B' es el gráfico de las setas.
       rst 16               ; poner la seta en la pantalla.
       djnz mushlp          ; bucle hasta que todas las setas aparezcan.

; Este es el bucle principal.

mloop  equ $

; Borrar el jugados.

       call basexy          ; establecer las posiciones x e y del jugador.
       call wspace          ; mostrar un espacio sobre el jugador.

; Ahora hemos eliminado el jugador y lo movemos antes de volverlo a mostrar
; en sus nuevas coordenadas.

       ld bc,63486          ; fila del teclado 1-5/joystick puerto 2.
       in a,(c)             ; ver que teclas están pulsadas.
       rra                  ; bit mas externo = tecla 1.
       push af              ; recordar el valor.
       call nc,mpl          ; si está siendo pulsada, moverse a la izquierda.
       pop af               ; restaurar el acumulador.
       rra                  ; siguiente bit (valor 2) = tecla 2.
       push af              ; recordar el valor.
       call nc,mpr          ; si está siendo pulsada, moverse a la derecha.
       pop af               ; restaurar el acumulador.
       rra                  ; siguiente bit (valor 4) = tecla 3.
       push af              ; recordar el valor.
       call nc,mpd          ; si está está siendo pulsada, moverse hacia abajo.
```

```
       pop af              ; restaurar el acumulador.
       rra                 ; siguiente bit (valor 8) = tecla 4.
       call nc,mpu         ; si está está siendo pulsada, moverse hacia arriba.

; Ahora que se ha movido podemos volver a mostrar al jugador.

       call basexy         ; establecer las posiciones x e y del jugador.
       call splayr         ; mostrar al jugador.

       halt                ; retardo.

; Saltar de nuevo al principio del bucle principal.

       jp mloop

; Mover al jugador a la izquierda.

mpl    ld hl,ply           ; recordar, ¡y es la coordenada horizontal!
       ld a,(hl)           ; ¿cuál es el valor actual?
       and a               ; ¿es cero?
       ret z               ; sí - no podemos seguir hacia la izquierda.
       dec (hl)            ; restar 1 a la coordenada y.
       ret

; Mover al jugador a la derecha.

mpr    ld hl,ply           ; recordar, ¡y es la coordenada horizontal!
       ld a,(hl)           ; ¿cuál es el valor actual?
       cp 31               ; ¿está en el borde derecho (31)?
       ret z               ; sí - no podemos seguir hacia la derecha.
       inc (hl)            ; sumar 1 a la coordenada y.
       ret

; Mover al jugador hacia arriba.

mpu    ld hl,plx           ; recordar, ¡x es la coordenada vertical!
       ld a,(hl)           ; ¿cuál es el valor actual?
       cp 4                ; ¿está en el límite superior (4)?
       ret z               ; sí - no podemos seguir hacia arriba.
       dec (hl)            ; restar 1 a la coordenada x.
       ret

; Mover al jugador hacia abajo.

mpd    ld hl,plx           ; recordar, ¡x es la coordenada vertical!
       ld a,(hl)           ; ¿cuál es el valor actual?
       cp 21               ; ¿está en el límite inferior (21)?
       ret z               ; sí - no podemos seguir hacia abajo.
       inc (hl)            ; sumar 1 a la coordenada x.
       ret

; Configurar las coordenadas X e Y de la posición de la base del jugador,
; se le llama antes de la visualización y supresión de la base.

basexy ld a,22             ; código para AT.
       rst 16
       ld a,(plx)          ; coordenada vertical del jugador.
       rst 16              ; fijar la posición vertical del jugador.
       ld a,(ply)          ; coordenada horizontal del jugador.
       rst 16              ; fijar la posición horizontal del jugador.
       ret
```

```
; Mostrar al jugador en la posición de impresión actual.

splayr ld a,69              ; tinta cían (5) en fondo negro (0),
                            ; brillante (64).
       ld (23695),a         ; establecer nuestros colores temporales de pantalla.
       ld a,144             ; código ASCII para el UDG 'A'.
       rst 16               ; dibujar jugador.
       ret

wspace ld a,71              ; tinta blanca (7) en fondo negro (0),
                            ; brillante (64).
       ld (23695),a         ; establecer nuestros colores temporales de pantalla.
       ld a,32              ; carácter de ESPACIO.
       rst 16               ; dibujar espacio.
       ret

; Sencillo generador de números seudo-aleatorio.
; Seguir un puntero a través de la ROM (a partir de una semilla),
; retornando el contenido del byte en esa posición.

random ld hl,(seed)         ; puntero
       ld a,h
       and 31               ; mantenerlo en los primeros 8Kb de ROM.
       ld h,a
       ld a,(hl)            ; Obtener el número "aleatorio" de esa ubicación.
       inc hl               ; Incrementar el puntero.
       ld (seed),hl
       ret
seed   defw 0

plx    defb 0               ; coordenada x del jugador.
ply    defb 0               ; coordenada y del jugador.

; gráficos UDG.

blocks defb 16,16,56,56,124,124,254,254    ; base del jugador.
       defb 24,126,255,255,60,60,60,60     ; seta.
```

Descarga el fuente desde aquí

Una vez ejecutado este listado se parece más a un juego de Centipede de lo que lo hacía antes, pero hay un problema importante. Las setas se distribuyen de forma aleatoria alrededor de la pantalla, pero el jugador puede moverse a través de ellas. Se requiere algún tipo de detección de colisiones para evitar que esto ocurra, lo que se cubrirá en el siguiente capítulo.

Capítulo 5: Detección sencilla de colisiones con el fondo

Encontrando atributos

Cualquiera que haya pasado tiempo programando en Sinclair BASIC puede recordar la función ATTR. Es una manera de detectar los atributos de color de cualquier celda de carácter de la pantalla, y aunque es complicado de captar para el programador en BASIC, es muy útil para la detección sencilla de colisiones. El método era tan útil, de hecho, que su equivalente en lenguaje máquina fue empleado en una serie de juegos comerciales, y es de gran utilidad para el programador novato del Spectrum.

Hay dos maneras de encontrar los parámetros de los atributos de color de una celda de carácter particular en el Spectrum. Una mirada rápida al desensamblado de la ROM del Spectrum revela una rutina en la dirección 9603 que va a hacer el trabajo por nosotros, o también podemos calcular la dirección de memoria por nosotros mismos.

La forma más sencilla de encontrar un valor de atributo es usar un par de rutinas de la ROM:

```
       ld bc,(ballx)        ; poner x,y en el par de registros bc.
       call 9603            ; llamada ROM, obtiene el atributo (c,b) en la pila.
       call 11733           ; poner los atributos en el acumulador.
```

Sin embargo, es mucho más rápido hacer el cálculo nosotros mismos. También es útil calcular la dirección de un atributo y no sólo su valor en caso de que también queramos escribirlo.

Calculando Direcciones de Atributos

A diferencia del torpe diseño de pixeles del Spectrum (***NdT***: *de esto se habla mas adelante, pero solo hay que ver como se van presentando las pantallas de carga de los juegos para hacerse una idea*), las céldas de color, ubicadas en las direcciones 22.528 a 23.295 ambas inclusive, están dispuestas secuencialmente en la memoria RAM tal y como cabría esperar. En otras palabras, las céldas de atributo de la primera línea de la pantalla se encuentran en las direcciones 22528 a 22559 y van de izquierda a derecha, la segunda fila de celdas de colores van desde 22560 a 22591, y así sucesivamente. Para encontrar la dirección de una celda de color en la posición de impresión (**x**, **y**) sólo tenemos que multiplicar **x** por 32, sumarle **y**, finalizando con añadir 22528 al resultado. Al examinar el contenido de esta dirección podemos encontrar los colores que se muestran en una posición particular y actuar en consecuencia. En el siguiente ejemplo se calcula la dirección de un atributo en la posición de carácter (b, c) y lo devuelve en el par de registro **HL**.

```
; Calcular la dirección del atributo de carácter en (b,c).
```

```
atadd  ld a,b              ; posición x.
       rrca                ; multiplicar por 32.
       rrca
       rrca
       ld l,a              ; guardar también en l.
       and 3               ; enmascarar bits para byte alto.
       add a,88            ; 88*256=22528, inicio de atributos.
       ld h,a              ; byte alto completado.
       ld a,l              ; obtener x*32 de nuevo.
       and 224             ; enmascarar el byte bajo.
       ld l,a              ; guardar en l.
       ld a,c              ; obtener desplazamiento de y.
       add a,l             ; añadir al byte bajo.
       ld l,a              ; hl=Dirección de atributos.
       ld a,(hl)           ; devolver atributo en a.
       ret
```

Al mirar el contenido del byte en **HL** nos dará el valor del atributo, mientras que la escritura en la posición de memoria apuntada por **HL** cambiará el color en ese lugar.

Para dar sentido a los resultados, tenemos que saber que cada atributo se compone de 8 bits que están dispuestos de esta manera:

```
d0-d2  color de la tinta 0-7, 0=negro, 1=azul, 2=rojo,     3=magenta,
                              4=verde, 5=cian, 6=amarillo, 7=blanco
d3-d5  color del fondo 0-7,   0=negro, 1=azul, 2=rojo,     3=magenta,
                              4=verde, 5=cian, 6=amarillo, 7=blanco
d6     brillo,                0=normal, 1=alto
d7     intermitencia,         0=no, 1=intermitente
```

Mirar si el fondo es verde por ejemplo, podría implicar

```
       and 56              ; enmascarar todos los bits menos el fondo.
       cp 32               ; ¿es verde (4) * 8?
       jr z,green          ; Sí, hacer lo que sea cuando es verde.
```

mientras que la comprobación de si la tinta es color amarillo se podría hacer de esta manera

```
       and 7               ; sólo queremos los bits de la tinta.
       cp 6                ; ¿es de color amarillo (6)?
       jr z,yellow         ; Sí, hacer lo que sea cuando es amarillo .
```

Aplicando lo que hemos aprendido al Juego

Ahora podemos añadir una prueba de colisión por atributo a nuestro juego del ciempiés. Como antes, las nuevas secciones están remarcadas.

```
; Queremos una pantalla en negro.

       ld a,71              ; tinta blanca (7) en fondo negro (0),
                            ; con brillo (64).
       ld (23693),a         ; establecer nuestros colores de pantalla.
       xor a                ; forma rápida de cargar el acumulador con cero.
       call 8859            ; establecer el colore del borde permanente.

; Configurar los gráficos.

       ld hl,blocks         ; dirección de los datos para los UDG.
       ld (23675),hl        ; apuntar los UDG hacia aquí.

; De acuerdo, vamos a empezar el juego.

       call 3503            ; rutina ROM - borra la pantalla, abre el canal 2.

; Inicializar coordenadas.

       ld hl,21+15*256      ; cargar el par hl con las coordenadas iniciales.
       ld (plx),hl          ; fijar las coordenadas del jugador.

       call basexy          ; establecer las posiciones x e y del jugador.
       call splayr          ; mostrar símbolo de la base del jugador.

; Ahora queremos llenar la zona de juegos con setas.

       ld a,68              ; tinta verde (4) en fondo negro (0),
                            ; con brillo (64).
       ld (23695),a         ; establecer nuestros colores temporales.
       ld b,50              ; comenzar con unas pocas.
mushlp ld a,22              ; código del carácter de control para AT.
       rst 16
       call random          ; obtener un número 'aleatorio'.
       and 15               ; en vertical en rango de 0 a 15.
       rst 16
       call random          ; obtener otro número seudo-aleatorio.
       and 31               ; horizontal en el rango de 0 a 31.
       rst 16
       ld a,145             ; el UDG 'B' es el gráfico de las setas.
       rst 16               ; poner la seta en la pantalla.
       djnz mushlp          ; bucle hasta que todas las setas aparezcan.

; Este es el bucle principal.

mloop  equ $

; Borrar el jugados.

       call basexy          ; establecer las posiciones x e y del jugador.
       call wspace          ; mostrar un espacio sobre el jugador.

; Ahora hemos eliminado el jugador y lo movemos antes de volverlo a mostrar
; en sus nuevas coordenadas.

       ld bc,63486          ; fila del teclado 1-5/joystick puerto 2.
       in a,(c)             ; ver que teclas están pulsadas.
       rra                  ; bit mas externo = tecla 1.
       push af              ; recordar el valor.
       call nc,mpl          ; si está siendo pulsada, moverse a la izquierda.
       pop af               ; restaurar el acumulador.
```

```
       rra                  ; siguiente bit (valor 2) = tecla 2.
       push af              ; recordar el valor.
       call nc,mpr          ; si está siendo pulsada, moverse a la derecha.
       pop af               ; restaurar el acumulador.
       rra                  ; siguiente bit (valor 4) = tecla 3.
       push af              ; recordar el valor.
       call nc,mpd          ; si está siendo pulsada, moverse hacia abajo.
       pop af               ; restaurar el acumulador.
       rra                  ; siguiente bit (valor 8) = tecla 4.
       call nc,mpu          ; si está siendo pulsada, moverse hacia arriba.

; Ahora que se ha movido podemos volver a mostrar al jugador.

       call basexy          ; establecer las posiciones x e y del jugador.
       call splayr          ; mostrar al jugador.

       halt                 ; retardo.

; Saltar de nuevo al principio del bucle principal.

       jp mloop

; Mover al jugador a la izquierda.

mpl    ld hl,ply            ; recordar, ¡y es la coordenada horizontal!
       ld a,(hl)            ; ¿cuál es el valor actual?
       and a                ; ¿es cero?
       ret z                ; sí - no podemos seguir hacia la izquierda.

; antes comprobar que no hay una seta en el camino.

       ld bc,(plx)          ; coordenadas actuales.
       dec b                ; mirar una posición a la izquierda.
       call atadd           ; obtener la dirección del atributo en esta posición.
       cp 68                ; las setas son brillo (64) + verde (4).
       ret z                ; hay una seta, no podemos movernos aquí.

       dec (hl)             ; restar 1 a la coordenada y.
       ret

; Mover al jugador a la derecha.

mpr    ld hl,ply            ; recordar, ¡y es la coordenada horizontal!
       ld a,(hl)            ; ¿cuál es el valor actual?
       cp 31                ; ¿está en el borde derecho (31)?
       ret z                ; sí - no podemos seguir hacia la derecha.

; antes comprobar que no hay una seta en el camino.

       ld bc,(plx)          ; coordenadas actuales.
       inc b                ; mirar una posición a la derecha.
       call atadd           ; obtener la dirección del atributo en esta posición.
       cp 68                ; las setas son brillo (64) + verde (4).
       ret z                ; hay una seta, no podemos movernos aquí.

       inc (hl)             ; sumar 1 a la coordenada y.
       ret

; Mover al jugador hacia arriba.

mpu    ld hl,plx            ; recordar, ¡x es la coordenada vertical!
```

```
       ld a,(hl)           ; ¿cuál es el valor actual?
       cp 4                ; ¿está en el límite superior (4)?
       ret z               ; sí - no podemos seguir hacia arriba.

; antes comprobar que no hay una seta en el camino.

       ld bc,(plx)         ; coordenadas actuales.
       dec c               ; mirar una posición hacia arriba.
       call atadd          ; obtener la dirección del atributo en esta posición.
       cp 68               ; las setas son brillo (64) + verde (4).
       ret z               ; hay una seta, no podemos movernos aquí.

       dec (hl)            ; restar 1 a la coordenada x.
       ret

; Mover al jugador hacia abajo.

mpd    ld hl,plx           ; recordar, ¡x es la coordenada vertical!
       ld a,(hl)           ; ¿cuál es el valor actual?
       cp 21               ; ¿está en el límite inferior (21)?
       ret z               ; sí - no podemos seguir hacia abajo.

; antes comprobar que no hay una seta en el camino.

       ld bc,(plx)         ; coordenadas actuales.
       inc c               ; mirar una posición hacia abajo.
       call atadd          ; obtener la dirección del atributo en esta posición.
       cp 68               ; las setas son brillo (64) + verde (4).
       ret z               ; hay una seta, no podemos movernos aquí.

       inc (hl)            ; sumar 1 a la coordenada x.
       ret

; Configurar las coordenadas X e Y de la posición de la base del jugador,
; se le llama antes de la visualización y supresión de la base.

basexy ld a,22             ; código para AT.
       rst 16
       ld a,(plx)          ; coordenada vertical del jugador.
       rst 16              ; fijar la posición vertical del jugador.
       ld a,(ply)          ; coordenada horizontal del jugador.
       rst 16              ; fijar la posición horizontal del jugador.
       ret

; Mostrar al jugador en la posición de impresión actual.

splayr ld a,69             ; tinta cían (5) en fondo negro (0),
                           ; brillante (64).
       ld (23695),a        ; establecer nuestros colores temporales de pantalla.
       ld a,144            ; código ASCII para el UDG 'A'.
       rst 16              ; dibujar jugador.
       ret

wspace ld a,71             ; tinta blanca (7) en fondo negro (0),
                           ; brillante (64).
       ld (23695),a        ; establecer nuestros colores temporales de pantalla.
       ld a,32             ; carácter de ESPACIO.
       rst 16              ; dibujar espacio.
       ret

; Sencillo generador de números seudo-aleatorio.
```

```
; Seguir un puntero a través de la ROM (a partir de una semilla),
; retornando el contenido del byte en esa posición.

random ld hl,(seed)         ; puntero
       ld a,h
       and 31               ; mantenerlo en los primeros 8Kb de ROM.
       ld h,a
       ld a,(hl)            ; Obtener el número "aleatorio" de esa ubicación.
       inc hl               ; Incrementar el puntero.
       ld (seed),hl
       ret
seed   defw 0

; Calcular la dirección del atributo de carácter en (dispx, dispy).

atadd  ld a,c               ; coordenada vertical.
       rrca                 ; multiplicar por 32.
       rrca                 ; desplazar a la derecha con acarreo 3 veces
       rrca                 ; mas rápido que desplazar izquierda 5 veces.
       ld e,a
       and 3
       add a,88             ; 88x256=dirección de los atributos.
       ld d,a
       ld a,e
       and 224
       ld e,a
       ld a,b               ; posición horizontal.
       add a,e
       ld e,a               ; de=dirección de los atributos.
       ld a,(de)            ; devolver atributo en el acumulador.
       ret

plx    defb 0               ; coordenada x del jugador.
ply    defb 0               ; coordenada y del jugador.

; gráficos UDG.

blocks defb 16,16,56,56,124,124,254,254    ; base del jugador.
       defb 24,126,255,255,60,60,60,60     ; seta.
```

Descarga el programa desde aquí

NdT*: Los cuatro bloques de código para detectar las colisiones son iguales salvo una instrucción, recomiendo reemplazarlos por una MACRO en lugar de repetir código, podía ser por ejemplo esto (la sintaxis exacta depende de su programa ensablador, este es el ejemplo en PASMO):*

```
MACRO  COLISION, INS
       ld bc,(plx)          ; coordenadas actuales.
       INS                  ; mirar una posición a la izquierda.
       call atadd           ; obtener la dirección del atributo en esta posición.
       cp 68                ; las setas son brillo (64) + verde (4).
       ret z                ; hay una seta, no podemos movernos aquí.
ENDM

......

; antes comprobar que no hay una seta en el camino.
```

```
        COLISION dec c
```

Capítulo 6: Tablas

Los aliens no llegan de uno en uno

Digamos, por poner un ejemplo, que estábamos escribiendo un juego de space invaders que muestra once columnas, cada una con cinco filas de invasores. No sería práctico escribir el código para cada uno de los cincuenta y cinco aliens en cada turno, lo que necesitamos es montar una tabla. En Sinclair BASIC podríamos hacerlo mediante la definición de tres tablas de cincuenta y cinco elementos (una para la coordenada **x** de los invasores, otra para la coordenada **y**, más una tercera con el estado). Podríamos hacer algo similar en ensamblador mediante la creación en memoria de tres tablas de cincuenta y cinco bytes cada una, y a continuación añadir el número de cada alien al inicio de cada tabla para acceder al elemento individual. Desafortunadamente, eso sería lento y engorroso.

Un método mucho mejor es agrupar los tres elementos de datos para cada invasor en una estructura, y luego montar cincuenta y cinco de estas estructuras en una tabla. Podemos entonces apuntar con **hl** a la dirección de cada invasor, y sabemos que **hl** apunta al byte de estado, **hl** más uno apunta a la coordenada **x**, **hl** mas dos apunta a la coordenada **y**. El código para mostrar un alien podría ser algo como esto:

```
       ld hl,aliens        ; estructura de datos para los aliens.
       ld b,55             ; número de aliens.
loop0  call show           ; presentar este alien.
       djnz loop0          ; repetir para todos los aliens.
       ret
show   ld a,(hl)           ; obtener el estado del alien.
       cp 255              ; ¿está el alien desactivado?
       jr z,next           ; si, entonces no presentarlo.
       push hl             ; guardar la dirección del alien en la pila.
       inc hl              ; apuntar a la coordenada x.
       ld d,(hl)           ; obtener la coordenada.
       inc hl              ; apuntar a la coordenada y.
       ld e,(hl)           ; obtener la coordenada.
       call disply         ; mostrar el alien en (d,e).
       pop hl              ; refrescar la dirección del alien desde la pila.
next   ld de,3             ; tamaño de cada alien en la tabla de entradas.
       add hl,de           ; apuntar al siguiente alien.
       ret                 ; mantiene hl apuntando al siguiente.
```

Usando los registros índice

El inconveniente con esta rutina es que tenemos que tener mucho cuidado de a donde está apuntando **hl** todo el tiempo (***NdT**: se refiere a que usando este método no podemos ir a un elemento directamente, debemos recorrerlos todos secuancialmente*) por lo que podría ser una buena idea almacenar **hl** en una posición de la memoria temporal de dos bytes antes de llamar a **show**, y al final del bucle restaurarla sumandole tres, para a continuación realizar la instrucción **djnz**. Si estábamos escribiendo para la Nintendo Game Boy con su Z80 recortado, esta sería

probablemente nuestra mejor opción. En máquinas con procesadores más avanzados, como el Spectrum y CPC 464 podemos utilizar los registros índice **ix** para simplificar nuestro código un poco. Debido a que el par de registro **ix** nos permite desplazamos usando direccionamiento indirecto, podemos apuntar **ix** al comienzo de la estructura de datos de un alien y acceder a todos los elementos en ella sin la necesidad de cambiar de nuevo **ix**. Usando **ix** nuestra rutina de visualización de aliens podría tener este aspecto:

```
       ld ix,aliens         ; estructura de datos para los aliens.
       ld b,55              ; número de aliens.
loop0  call show            ; presentar este alien.
       ld de,3              ; tamaño de cada alien en la tabla de entradas.
       add ix,de            ; apuntar al siguiente alien.
       djnz loop0           ; repetir para todos los aliens.
       ret
show   ld a,(ix)            ; obtener el estado del alien.
       cp 255               ; ¿está el alien desactivado?
       ret z                ; si, entonces no presentarlo.
       ld d,(ix+1)          ; obtener la coordenada.
       ld e,(ix+2)          ; obtener la coordenada.
       jp disply            ; mostrar el alien en (d,e).
```

Usar **ix** significa que sólo la primera vez tienes que señalar el comienzo de la estructura de datos de un alien, por lo que siempre **ix + 0** podrá devolver el estado para el invasor actual, **ix + 1** la coordenada **x**, y así sucesivamente. Este método permite al programador utilizar estructuras de datos complejas para sus aliens de hasta 128 bytes de largo, sin confundirse en cuanto de a que punto de la estructura de nuestros registros se está apuntando en un momento dado, como en el ejemplo anterior con **hl**. Desafortunadamente, el uso de **ix** es un poco más lento que **hl**, por lo que no se debe utilizar para las tareas de procesamiento más intensivas, como la manipulación de gráficos.

Vamos a aplicar este método a nuestro juego del Centipede. En primer lugar, tenemos que decidir cuantos segmentos se necesitan, y qué datos almacenar sobre cada segmento. En nuestro juego los segmentos tendrán que desplazarse hacia la izquierda o hacia la derecha hasta que lleguen a una seta, y luego moverse hacia abajo y volver a la inversa. Así que parece que necesitaremos una variable bandera para indicar la dirección en que un segmento está viajando, además de una coordenada **x** o **y**. Nuestra variable también se puede utilizar para indicar que un segmento particular ha sido destruido. Con esto en mente, podemos establecer una estructura de datos de tres bytes:

```
centf  defb 0               ; flag, 0=izquierda, 1=derecha, 255=muerto.
centx  defb 0               ; coordenada x del segmento.
centy  defb 0               ; coordenada y del segmento.
```

Si optamos por tener diez segmentos en nuestro Centipede, hay que reservar espacio para una tabla de treinta bytes. Cada segmento tiene que ser inicializado al principio del juego, para después moverlo, mostrarlo y eliminarlo durante el juego.

La inicialización de los segmentos es probablemente la tarea más simple, por lo que podemos utilizar un simple bucle para incrementar el par de registros **HL** para que apunte a cada byte antes de ajustarlo. Algo parecido a lo esto se usa en este truco:

```
       ld b,10            ; número de segmentos a inicializar.
       ld hl,segmnt       ; tabla de segmentos.
segint ld (hl),1          ; comienza moviéndose a la derecha.
       inc hl
       ld (hl),0          ; comienza arriba.
       inc hl
       ld (hl),b          ; usa el registro B para la coordenada y.
       inc hl
       djnz segint        ; repetir hasta que todos se inicialicen.
```

El procesamiento y la visualización de cada segmento va a ser un poco más complicado, por lo que para eso vamos a utilizar los registros **ix**. Necesitamos escribir un algoritmo simple que manipula un solo segmento hacia la izquierda o hacia la derecha hasta que llega a una seta, y luego se mueve hacia abajo y cambia de dirección. Llamaremos a esta rutina **PROSEG** (por "procesar segmento"), y establecer un bucle que apunte a su vez a cada segmento y llame a **PROSEG**. Proporcionando el correcto algoritmo de movimiento, podremos ver a continuación un ciempiés que serpentea su camino a través de las setas. Aplicar esto a nuestro código es sencillo: comprobamos el byte de bandera para cada segmento (ix) para ver de qué manera el segmento se está moviendo, incremento o decremento según la dirección de la coordenada horizontal (ix + 2), a continuación comprobar el atributo en esa celda de carácter. Si es verde y negro incrementamos la coordenada vertical (ix + 1) y cambiamos el indicador de dirección (ix).

De acuerdo, hay algunas cosas más a considerar, como cuando golpea contra los lados o el fondo de la pantalla, pero eso es sólo un caso mas al comprobar las coordenadas del segmento y cambiar su dirección, o su traslado a la parte superior de la pantalla cuando sea necesario. Los segmentos también necesitan ser borrados de sus antiguas posiciones antes de ser trasladados y mostrarlos en sus nuevas posiciones, pero ya se han cubierto los pasos necesarios para realizar esas tareas.

Nuestro nuevo código es el siguiente:

```
; Queremos una pantalla en negro.

       ld a,71            ; tinta blanca (7) en fondo negro (0),
                          ; con brillo (64).
       ld (23693),a       ; establecer nuestros colores de pantalla.
       xor a              ; forma rápida de cargar el acumulador con cero.
       call 8859          ; establecer el colore del borde permanente.

; Configurar los gráficos.

       ld hl,blocks       ; dirección de los datos para los UDG.
       ld (23675),hl      ; apuntar los UDG hacia aquí.
```

```
; De acuerdo, vamos a empezar el juego.

       call 3503           ; rutina ROM - borra la pantalla, abre el canal 2.

; Inicializar coordenadas.

       ld hl,21+15*256     ; cargar el par hl con las coordenadas iniciales.
       ld (plx),hl         ; fijar las coordenadas del jugador.

       ld b,10             ; número de segmentos a inicializar.
       ld hl,segmnt        ; tabla de segmentos.
segint ld (hl),1           ; comienza moviéndose a la derecha.
       inc hl
       ld (hl),0           ; comienza arriba.
       inc hl
       ld (hl),b           ; usa el registro B para la coordenada y.
       inc hl
       djnz segint         ; repetir hasta que todos se inicialicen.

       call basexy         ; establecer las posiciones x e y del jugador.
       call splayr         ; mostrar símbolo de la base del jugador.

; Ahora queremos llenar la zona de juegos con setas.

       ld a,68             ; tinta verde (4) en fondo negro (0),
                           ; con brillo (64).
       ld (23695),a        ; establecer nuestros colores temporales.
       ld b,50             ; comenzar con unas pocas.
mushlp ld a,22             ; código del carácter de control para AT.
       rst 16
       call random         ; obtener un número 'aleatorio'.
       and 15              ; en vertical en rango de 0 a 15.
       rst 16
       call random         ; obtener otro número seudo-aleatorio.
       and 31              ; horizontal en el rango de 0 a 31.
       rst 16
       ld a,145            ; el UDG 'B' es el gráfico de las setas.
       rst 16              ; poner la seta en la pantalla.
       djnz mushlp         ; bucle hasta que todas las setas aparezcan.

; Este es el bucle principal.

mloop  equ $

; Borrar el jugados.

       call basexy         ; establecer las posiciones x e y del jugador.
       call wspace         ; mostrar un espacio sobre el jugador.

; Ahora hemos eliminado el jugador y lo movemos antes de volverlo a mostrar
; en sus nuevas coordenadas.

       ld bc,63486         ; fila del teclado 1-5/joystick puerto 2.
       in a,(c)            ; ver que teclas están pulsadas.
       rra                 ; bit mas externo = tecla 1.
       push af             ; recordar el valor.
       call nc,mpl         ; si está siendo pulsada, moverse a la izquierda.
       pop af              ; restaurar el acumulador.
       rra                 ; siguiente bit (valor 2) = tecla 2.
       push af             ; recordar el valor.
       call nc,mpr         ; si está siendo pulsada, moverse a la derecha.
```

```
       pop af              ; restaurar el acumulador.
       rra                 ; siguiente bit (valor 4) = tecla 3.
       push af             ; recordar el valor.
       call nc,mpd         ; si está siendo pulsada, moverse hacia abajo.
       pop af              ; restaurar el acumulador.
       rra                 ; siguiente bit (valor 8) = tecla 4.
       call nc,mpu         ; si está siendo pulsada, moverse hacia arriba.

; Ahora que se ha movido podemos volver a mostrar al jugador.

       call basexy         ; establecer las posiciones x e y del jugador.
       call splayr         ; mostrar al jugador.

; Ahora los segmentos del ciempiés.

       ld ix,segmnt        ; tabla de datos del segmento.
       ld b,10             ; número de segmentos en la tabla.
censeg push bc
       ld a,(ix)           ; ¿está el segmento activado?
       inc a               ; 255=desactivado, incrementar a cero.
       call nz,proseg      ; si está activo, procesar el segmento.
       pop bc
       ld de,3             ; 3 bytes por segmento.
       add ix,de           ; poner el nuevo segmento en el registro ix.
       djnz censeg         ; repetir para todos los segmentos.

       halt                ; retardo.

; Saltar de nuevo al principio del bucle principal.

       jp mloop

; Mover al jugador a la izquierda.

mpl    ld hl,ply           ; recordar, ¡y es la coordenada horizontal!
       ld a,(hl)           ; ¿cuál es el valor actual?
       and a               ; ¿es cero?
       ret z               ; sí - no podemos seguir hacia la izquierda.

; antes comprobar que no hay una seta en el camino.

       ld bc,(plx)         ; coordenadas actuales.
       dec b               ; mirar una posición a la izquierda.
       call atadd          ; obtener la dirección del atributo en esta posición.
       cp 68               ; las setas son brillo (64) + verde (4).
       ret z               ; hay una seta, no podemos movernos aquí.

       dec (hl)            ; restar 1 a la coordenada y.
       ret

; Mover al jugador a la derecha.

mpr    ld hl,ply           ; recordar, ¡y es la coordenada horizontal!
       ld a,(hl)           ; ¿cuál es el valor actual?
       cp 31               ; ¿está en el borde derecho (31)?
       ret z               ; sí - no podemos seguir hacia la derecha.

; antes comprobar que no hay una seta en el camino.

       ld bc,(plx)         ; coordenadas actuales.
       inc b               ; mirar una posición a la derecha.
```

```
       call atadd          ; obtener la dirección del atributo en esta posición.
       cp 68               ; las setas son brillo (64) + verde (4).
       ret z               ; hay una seta, no podemos movernos aquí.

       inc (hl)            ; sumar 1 a la coordenada y.
       ret

; Mover al jugador hacia arriba.

mpu    ld hl,plx           ; recordar, ¡x es la coordenada vertical!
       ld a,(hl)           ; ¿cuál es el valor actual?
       cp 4                ; ¿está en el límite superior (4)?
       ret z               ; sí - no podemos seguir hacia arriba.

; antes comprobar que no hay una seta en el camino.

       ld bc,(plx)         ; coordenadas actuales.
       dec c               ; mirar una posición hacia arriba.
       call atadd          ; obtener la dirección del atributo en esta posición.
       cp 68               ; las setas son brillo (64) + verde (4).
       ret z               ; hay una seta, no podemos movernos aquí.

       dec (hl)            ; restar 1 a la coordenada x.
       ret

; Mover al jugador hacia abajo.

mpd    ld hl,plx           ; recordar, ¡x es la coordenada vertical!
       ld a,(hl)           ; ¿cuál es el valor actual?
       cp 21               ; ¿está en el límite inferior (21)?
       ret z               ; sí - no podemos seguir hacia abajo.

; antes comprobar que no hay una seta en el camino.

       ld bc,(plx)         ; coordenadas actuales.
       inc c               ; mirar una posición hacia abajo.
       call atadd          ; obtener la dirección del atributo en esta posición.
       cp 68               ; las setas son brillo (64) + verde (4).
       ret z               ; hay una seta, no podemos movernos aquí.

       inc (hl)            ; sumar 1 a la coordenada x.
       ret

; Configurar las coordenadas X e Y de la posición de la base del jugador,
; se le llama antes de la visualización y supresión de la base.

basexy ld a,22             ; código para AT.
       rst 16
       ld a,(plx)          ; coordenada vertical del jugador.
       rst 16              ; fijar la posición vertical del jugador.
       ld a,(ply)          ; coordenada horizontal del jugador.
       rst 16              ; fijar la posición horizontal del jugador.
       ret

; Mostrar al jugador en la posición de impresión actual.

splayr ld a,69             ; tinta cían (5) en fondo negro (0),
                           ; brillante (64).
       ld (23695),a        ; establecer nuestros colores temporales de pantalla.
       ld a,144            ; código ASCII para el UDG 'A'.
       rst 16              ; dibujar jugador.
```

```
       ret

wspace ld a,71             ; tinta blanca (7) en fondo negro (0),
                           ; brillante (64).
       ld (23695),a        ; establecer nuestros colores temporales de pantalla.
       ld a,32             ; carácter de ESPACIO.
       rst 16              ; dibujar espacio.
       ret

segxy  ld a,22             ; código ASCII para el carácter de AT.
       rst 16              ; presentar código AT.
       ld a,(ix+1)         ; obtener la coordenada x del segmento.
       rst 16              ; posicionarse en esa coordenada.
       ld a,(ix+2)         ; obtener la coordenada y del segmento.
       rst 16              ; posicionarse en esa coordenada.
       ret

proseg ld a,(ix)           ; verificar si el segmento está activo.
       inc a               ; para la rutina de detección de colisiones.
       ret z               ; está activo, por tanto pasa a muerto.
       call segxy          ; actualizar las coordenadas del segmento.
       call wspace         ; presentar un espacio, blanco sobre fondo negro.
       call segmov         ; mover el segmento.
       ld a,(ix)           ; verificar si el segmento está activado.
       inc a               ; para la rutina de detección de colisiones.
       ret z               ; está activo, por tanto pasa a muerto.
       call segxy          ; actualizar las coordenadas del segmento.
       ld a,2              ; código de atributo = 2, segmento rojo.
       ld (23695),a        ; actualizar los atributos temporales.
       ld a,146            ; UDG 'C' para presentar el segmento.
       rst 16
       ret
segmov ld a,(ix+1)         ; coordenada x.
       ld c,a              ; área x GP.
       ld a,(ix+2)         ; coordenada y.
       ld b,a              ; área y GP.
       ld a,(ix)           ; indicador de estado.
       and a               ; ¿está el segmento en el borde izquierdo?
       jr z,segml          ; ir a la izquierda, saltar según ese bit de código.

; ¡ahora el segmento se mueve a la derecha!

segmr  ld a,(ix+2)         ; coordenada y.
       cp 31               ; ¿está en el borde derecho de la pantalla?
       jr z,segmd          ; si, mover el segmento hacia abajo.
       inc a               ; ver la izquierda.
       ld b,a              ; actualizar la coordenada y GP.
       call atadd          ; mirar el atributo de esa dirección.
       cp 68               ; setas son brillo (64) + verde (4).
       jr z,segmd          ; seta a la derecha, moverse hacia abajo.
       inc (ix+2)          ; sin obstáculos, seguir hacia la derecha.
       ret

; ¡ahora el segmento se mueve a la izquierda!

segml  ld a,(ix+2)         ; coordenada y.
       and a               ; ¿está en el borde izquierdo de la pantalla?
       jr z,segmd          ; si, mover el segmento hacia abajo.
       dec a               ; ver la derecha.
       ld b,a              ; actualizar la coordenada y GP.
       call atadd          ; mirar el atributo en la dirección (dispx,dispy).
```

```
       cp 68                 ; setas son brillo (64) + verde (4).
       jr z,segmd            ; seta a la izquierda, moverse abajo.
       dec (ix+2)            ; sin obstáculos, seguir a la izquierda.
       ret

; ¡ahora el segmento se mueve hacia abajo!

segmd  ld a,(ix)             ; dirección del segmento.
       xor 1                 ; cambiarla.
       ld (ix),a             ; guardar la nueva dirección.
       ld a,(ix+1)           ; coordenada y.
       cp 21                 ; ¿llegamos al final de la pantalla?
       jr z,segmt            ; si, mover el segmento al inicio.


; En este momento nos estamos moviendo hacia abajo independientemente de las
; setas que pueden bloquear la trayectoria del segmento. Cualquier cosa en el
; camino del segmento será borrada.

       inc (ix+1)            ; no ha llegado a la parte inferior, bajar.
       ret

; mover segmento a la parte superior de la pantalla.

segmt  xor a                 ; igual que ld a,0 pero ahorra 1 byte.
       ld (ix+1),a           ; nueva coordenada x = inicio de la pantalla.
       ret


; Sencillo generador de números seudo-aleatorio.
; Seguir un puntero a través de la ROM (a partir de una semilla),
; retornando el contenido del byte en esa posición.

random ld hl,(seed)          ; puntero
       ld a,h
       and 31                ; mantenerlo en los primeros 8Kb de ROM.
       ld h,a
       ld a,(hl)             ; Obtener el número "aleatorio" de esa ubicación.
       inc hl                ; Incrementar el puntero.
       ld (seed),hl
       ret
seed   defw 0

; Calcular la dirección del atributo de carácter en (dispx, dispy).

atadd  ld a,c                ; coordenada vertical.
       rrca                  ; multiplicar por 32.
       rrca                  ; desplazar a la derecha con acarreo 3 veces
       rrca                  ; mas rápido que desplazar izquierda 5 veces.
       ld e,a
       and 3
       add a,88              ; 88x256=dirección de los atributos.
       ld d,a
       ld a,e
       and 224
       ld e,a
       ld a,b                ; posición horizontal.
       add a,e
       ld e,a                ; de=dirección de los atributos.
       ld a,(de)             ; devolver atributo en el acumulador.
       ret
```

```
plx    defb 0              ; coordenada x del jugador.
ply    defb 0              ; coordenada y del jugador.

; gráficos UDG.

blocks defb 16,16,56,56,124,124,254,254    ; base del jugador.
       defb 24,126,255,255,60,60,60,60     ; seta.
       defb 24,126,126,255,255,126,126,24  ; segmento.

; Tabla de segmentos.
; Formato: 3 bytes por entrada, 10 segmentos.
; byte 1: 255=segmento apagado, 0=izquierda, 1=derecha.
; byte 2: coordenada x (vertical).
; byte 3: coordenada y (horizontal).

segmnt defb 0,0,0          ; segmento 1.
       defb 0,0,0          ; segmento 2.
       defb 0,0,0          ; segmento 3.
       defb 0,0,0          ; segmento 4.
       defb 0,0,0          ; segmento 5.
       defb 0,0,0          ; segmento 6.
       defb 0,0,0          ; segmento 7.
       defb 0,0,0          ; segmento 8.
       defb 0,0,0          ; segmento 9.
       defb 0,0,0          ; segmento 10.
```

Descarga el programa desde aquí

***NdT**: El direccionamiento indexado necesita sumar siempre una cantidad al puntero, por lo que si no se indica será cero, pero algunos ensambladores no soportan la instrucción **(ix)** a secas, debes usar en su lugar **(ix+0)**, sería poner por ejemplo lo siguiente:*

```
segmd  ld a,(ix+0)         ; dirección del segmento.
       xor 1               ; cambiarla.
       ld (ix+0),a         ; guardar la nueva dirección.
       ld a,(ix+1)         ; coordenada y.
       cp 21               ; ¿llegamos al final de la pantalla?
       jr z,segmt          ; si, mover el segmento al inicio.
```

Capítulo 7: Detección simple de colisiones con aliens

Comprobación de coordenadas

La comprobación de coordenadas deberían poder explicarla por sí mismo la mayoría de los programadores, pero se ha incluido aquí en aras de la exhaustividad. También es el siguiente paso en el desarrollo de nuestro juego del ciempiés.

El tipo más simple de detección de colisión sería algo como esto, que se utiliza para detectar si dos UDG han chocado:

```
       ld a,(playx)        ; coordenada x del jugador.
       cp (ix+1)           ; comparar con x del alien.
       ret nz              ; no es la misma, no hay colisión.
       ld a,(playy)        ; coordenada y del jugador.
       cp (ix+2)           ; comparar con y del alien.
       ret nz              ; no es la misma, no hay colisión.
       jp collis           ; tenemos una colisión.
```

Bueno, es bastante simple, pero la mayoría de los juegos no utilizan una sola celda de carácter para los gráficos. ¿Qué pasaría si los aliens son de cuatro celdas de caracteres de ancho por dos de alto, y el personaje del jugador es de tres celdas de alto por tres de ancho? Tenemos que comprobar si alguna parte del alien ha colisionado con cualquier parte del jugador, por lo que tenemos que comprobar que las coordenadas están dentro de un cierto rango. Si el alien está más cerca de dos cuadrados por encima del jugador o menos de 3 por debajo, entonces las coordenadas verticales coinciden. Si el alien está también a menos de cuatro celdas a la izquierda del jugador, y menos de tres celdas a la derecha, la posición horizontal también se comparte y tenemos una colisión.

Vamos a escribir algo de código para hacer esto. Podemos empezar por tomar la coordenada vertical del jugador:

```
       ld a,(playx)        ; coordenada x del jugador.
```

Luego restar la posición vertical del alien:

```
       sub (ix+1)          ; restar la x del alien.
```

A continuación, resta uno de la altura del jugador y sumárselo.

```
       add a,2              ; el jugador es de 3 de alto, luego añadir 3 - 1 = 2
```

Si el alien se encuentra dentro del rango, el resultado será menor que la altura combinada del jugador y el alien, por lo que realizamos la comprobación:

```
       cp 5                 ; la altura combinada es 3 + 2 = 5.
       ret nc               ; no en el rango vertical.
```

De forma similar, podemos seguir este código para comprobar la horizontal:

```
       ld a,(playy)         ; coordenada y del jugador.
       sub (ix+2)           ; restar la y del alien.
       add a,2              ; el jugador es de 3 de ancho, luego añadir 3 - 1 = 2
       cp 7                 ; el ancho combinado es 3 + 4 = 7.
       ret nc               ; no en el rango horizontal.
       jp collis            ; tenemos una colisión.
```

Por supuesto, este método no solo trabaja con gráficos basados en caracteres, también funciona perfectamente bien con los sprites, pero eso lo veremos más adelante. Es hora de **FINALIZAR** nuestro juego del Centipede con un poco de detección de colisiones. Como nuestros gráficos son todos UDG de un solo carácter no necesitamos mucha fantasía, una rápida verificación de si **x** = **x** y de si **y** = **y** es lo que necesitamos.

```
numseg equ 8               ; numero de segmentos del ciempiés.

; Queremos una pantalla en negro.

       ld a,71              ; tinta blanca (7) en fondo negro (0),
                            ; con brillo (64).
       ld (23693),a         ; establecer nuestros colores de pantalla.
       xor a                ; forma rápida de cargar el acumulador con cero.
       call 8859            ; establecer el colore del borde permanente.

; Configurar los gráficos.

       ld hl,blocks         ; dirección de los datos para los UDG.
       ld (23675),hl        ; apuntar los UDG hacia aquí.

; De acuerdo, vamos a empezar el juego.

       call 3503            ; rutina ROM - borra la pantalla, abre el canal 2.

       xor a                ; poner a cero el acumulador.
       ld (dead),a          ; poner el flag que indica que el jugador está vivo.

; Inicializar coordenadas.

       ld hl,21+15*256      ; cargar el par hl con las coordenadas iniciales.
       ld (plx),hl          ; fijar las coordenadas del jugador.
       ld hl,255+255*256    ; disparo del jugador por defecto.
       ld (pbx),hl          ; poner las coordenadas del disparo.
```

```
       ld b,10              ; número de segmentos a inicializar.
       ld hl,segmnt         ; tabla de segmentos.
segint ld (hl),1            ; comienza moviéndose a la derecha.
       inc hl
       ld (hl),0            ; comienza arriba.
       inc hl
       ld (hl),b            ; usa el registro B para la coordenada y.
       inc hl
       djnz segint          ; repetir hasta que todos se inicialicen.

       call basexy          ; establecer las posiciones x e y del jugador.
       call splayr          ; mostrar símbolo de la base del jugador.

; Ahora queremos llenar la zona de juegos con setas.

       ld a,68              ; tinta verde (4) en fondo negro (0),
                            ; con brillo (64).
       ld (23695),a         ; establecer nuestros colores temporales.
       ld b,50              ; comenzar con unas pocas.
mushlp ld a,22              ; código del carácter de control para AT.
       rst 16
       call random          ; obtener un número 'aleatorio'.
       and 15               ; en vertical en rango de 0 a 15.
       rst 16
       call random          ; obtener otro número seudo-aleatorio.
       and 31               ; horizontal en el rango de 0 a 31.
       rst 16
       ld a,145             ; el UDG 'B' es el gráfico de las setas.
       rst 16               ; poner la seta en la pantalla.
       djnz mushlp          ; bucle hasta que todas las setas aparezcan.

; Este es el bucle principal.

mloop  equ $

; Borrar el jugados.

       call basexy          ; establecer las posiciones x e y del jugador.
       call wspace          ; mostrar un espacio sobre el jugador.

; Ahora hemos eliminado el jugador y lo movemos antes de volverlo a mostrar
; en sus nuevas coordenadas.

       ld bc,63486          ; fila del teclado 1-5/joystick puerto 2.
       in a,(c)             ; ver que teclas están pulsadas.
       rra                  ; bit mas externo = tecla 1.
       push af              ; recordar el valor.
       call nc,mpl          ; si está siendo pulsada, moverse a la izquierda.
       pop af               ; restaurar el acumulador.
       rra                  ; siguiente bit (valor 2) = tecla 2.
       push af              ; recordar el valor.
       call nc,mpr          ; si está siendo pulsada, moverse a la derecha.
       pop af               ; restaurar el acumulador.
       rra                  ; siguiente bit (valor 4) = tecla 3.
       push af              ; recordar el valor.
       call nc,mpd          ; si está siendo pulsada, moverse hacia abajo.
       pop af               ; restaurar el acumulador.
       rra                  ; siguiente bit (valor 8) = tecla 4.
       push af              ; recordar este valor.
       call nc,mpu          ; si está siendo pulsada, moverse hacia arriba.
```

```
       pop af              ; restaurar el acumulador.
       rra                 ; ultimo bit (valor 16) leer tecla 5.
       call nc,fire        ; se ha pulsado, moverse arriba.

; Ahora que se ha movido podemos volver a mostrar al jugador.

       call basexy         ; establecer las posiciones x e y del jugador.
       call splayr         ; mostrar al jugador.

; Ahora para el disparo.  Primero vemos si ha golpeado algo.

       call bchk           ; ver la posición del disparo.

       call dbull          ; borrar disparo.
       call moveb          ; mover el disparo.
       call bchk           ; calcular nueva posición del disparo.
       call pbull          ; presentar el disparo en su nueva posición.

; Ahora los segmentos del ciempiés.

       ld ix,segmnt        ; tabla de datos del segmento.
       ld b,10             ; número de segmentos en la tabla.
censeg push bc
       ld a,(ix)           ; ¿está el segmento activado?
       inc a               ; 255=desactivado, incrementar a cero.
       call nz,proseg      ; si está activo, procesar el segmento.
       pop bc
       ld de,3             ; 3 bytes por segmento.
       add ix,de           ; poner el nuevo segmento en el registro ix.
       djnz censeg         ; repetir para todos los segmentos.

       halt                ; retardo.

       ld a,(dead)         ; ¿ha muerto el jugador por un segmento?
       and a
       ret nz              ; jugador muerto, perder una vida.

; Saltar de nuevo al principio del bucle principal.

       jp mloop

; Mover al jugador a la izquierda.

mpl    ld hl,ply           ; recordar, ¡y es la coordenada horizontal!
       ld a,(hl)           ; ¿cuál es el valor actual?
       and a               ; ¿es cero?
       ret z               ; sí - no podemos seguir hacia la izquierda.

; antes comprobar que no hay una seta en el camino.

       ld bc,(plx)         ; coordenadas actuales.
       dec b               ; mirar una posición a la izquierda.
       call atadd          ; obtener la dirección del atributo en esta posición.
       cp 68               ; las setas son brillo (64) + verde (4).
       ret z               ; hay una seta, no podemos movernos aquí.

       dec (hl)            ; restar 1 a la coordenada y.
       ret

; Mover al jugador a la derecha.
```

```
mpr    ld hl,ply           ; recordar, ¡y es la coordenada horizontal!
       ld a,(hl)           ; ¿cuál es el valor actual?
       cp 31               ; ¿está en el borde derecho (31)?
       ret z               ; sí - no podemos seguir hacia la derecha.

; antes comprobar que no hay una seta en el camino.

       ld bc,(plx)         ; coordenadas actuales.
       inc b               ; mirar una posición a la derecha.
       call atadd          ; obtener la dirección del atributo en esta posición.
       cp 68               ; las setas son brillo (64) + verde (4).
       ret z               ; hay una seta, no podemos movernos aquí.

       inc (hl)            ; sumar 1 a la coordenada y.
       ret

; Mover al jugador hacia arriba.

mpu    ld hl,plx           ; recordar, ¡x es la coordenada vertical!
       ld a,(hl)           ; ¿cuál es el valor actual?
       cp 4                ; ¿está en el límite superior (4)?
       ret z               ; sí - no podemos seguir hacia arriba.

; antes comprobar que no hay una seta en el camino.

       ld bc,(plx)         ; coordenadas actuales.
       dec c               ; mirar una posición hacia arriba.
       call atadd          ; obtener la dirección del atributo en esta posición.
       cp 68               ; las setas son brillo (64) + verde (4).
       ret z               ; hay una seta, no podemos movernos aquí.

       dec (hl)            ; restar 1 a la coordenada x.
       ret

; Mover al jugador hacia abajo.

mpd    ld hl,plx           ; recordar, ¡x es la coordenada vertical!
       ld a,(hl)           ; ¿cuál es el valor actual?
       cp 21               ; ¿está en el límite inferior (21)?
       ret z               ; sí - no podemos seguir hacia abajo.

; antes comprobar que no hay una seta en el camino.

       ld bc,(plx)         ; coordenadas actuales.
       inc c               ; mirar una posición hacia abajo.
       call atadd          ; obtener la dirección del atributo en esta posición.
       cp 68               ; las setas son brillo (64) + verde (4).
       ret z               ; hay una seta, no podemos movernos aquí.

       inc (hl)            ; sumar 1 a la coordenada x.
       ret

; Disparar un misil.

fire   ld a,(pbx)          ; coordenada vertical del proyectil.
       inc a               ; 255 es el valor por defecto, se incrementa a cero.
       ret nz              ; proyectil en pantalla, no puede volver a disparar.
       ld hl,(plx)         ; coordenadas del jugador.
       dec l               ; l esquina superior.
       ld (pbx),hl         ; establece las coordenadas del proyectil.
       ret
```

```
bchk   ld a,(pbx)           ; proyectil vertical.
       inc a                ; ¿Está a 255 (defecto)?
       ret z                ; si, no hay proyectil en la pantalla.
       ld bc,(pbx)          ; obtiene las coordenadas.
       call atadd           ; buscar el atributo de aquí.
       cp 68                ; seta es brillo (64) + verde (4).
       jr z,hmush           ; ¡Alcanzada una seta!
       ret

hmush  ld a,22              ; código de control para AT.
       rst 16
       ld a,(pbx)           ; proyectil vertical.
       rst 16
       ld a,(pby)           ; proyectil horizontal.
       rst 16
       call wspace          ; establecer el color de la tinta a blanco.
kilbul ld a,255             ; coordenada x a 255 = apagar el proyectil.
       ld (pbx),a           ; destruir el proyectil.
       ret

; Mover el proyectil arriba de la pantalla 1 carácter a la vez.

moveb  ld a,(pbx)           ; proyectil vertical.
       inc a                ; ¿Está a 255 (defecto)?
       ret z                ; si, no hay proyectil en la pantalla.
       sub 2                ; subir una línea.
       ld (pbx),a
       ret

; Configurar las coordenadas X e Y de la posición de la base del jugador,
; se le llama antes de la visualización y supresión de la base.

basexy ld a,22              ; código para AT.
       rst 16
       ld a,(plx)           ; coordenada vertical del jugador.
       rst 16               ; fijar la posición vertical del jugador.
       ld a,(ply)           ; coordenada horizontal del jugador.
       rst 16               ; fijar la posición horizontal del jugador.
       ret

; Mostrar al jugador en la posición de impresión actual.

splayr ld a,69              ; tinta cían (5) en fondo negro (0),
                            ; brillante (64).
       ld (23695),a         ; establecer nuestros colores temporales de pantalla.
       ld a,144             ; código ASCII para el UDG 'A'.
       rst 16               ; dibujar jugador.
       ret

pbull  ld a,(pbx)           ; proyectil vertical.
       inc a                ; ¿Está a 255 (defecto)?
       ret z                ; si, no hay proyectil en la pantalla.
       call bullxy
       ld a,16              ; carácter de control para la tinta.
       rst 16
       ld a,6               ; 6 = amarillo.
       rst 16
       ld a,147             ; el UDG 'D' se usa para presentar el disparo.
       rst 16
       ret
```

```
dbull  ld a,(pbx)          ; proyectil vertical.
       inc a               ; ¿Está a 255 (defecto)?
       ret z               ; si, no hay proyectil en la pantalla.
       call bullxy         ; establece las coordenadas del proyectil.

wspace ld a,71             ; tinta blanca (7) en fondo negro (0),
                           ; brillante (64).
       ld (23695),a        ; establecer nuestros colores temporales de pantalla.
       ld a,32             ; carácter de ESPACIO.
       rst 16              ; dibujar espacio.
       ret

; Establece las coordenadas x e y de la posición del  proyectil del jugador,
; se llama a esta rutina antes de presentar y borrar el proyectil.

bullxy ld a,22             ; código para AT.
       rst 16
       ld a,(pbx)          ; proyectil del jugador coordenada vertical.
       rst 16              ; establece la posición vertical del jugador.
       ld a,(pby)          ; posición horizontal del proyectil.
       rst 16              ; establece la coordenada horizontal.
       ret

segxy  ld a,22             ; código ASCII para el carácter de AT.
       rst 16              ; presentar código AT.
       ld a,(ix+1)         ; obtener la coordenada x del segmento.
       rst 16              ; posicionarse en esa coordenada.
       ld a,(ix+2)         ; obtener la coordenada y del segmento.
       rst 16              ; posicionarse en esa coordenada.
       ret

proseg call segcol         ; detección de colisión con un segmento.
       ld a,(ix)           ; verificar si el segmento está activo.
       inc a               ; para la rutina de detección de colisiones.
       ret z               ; está activo, por tanto pasa a muerto.
       call segxy          ; actualizar las coordenadas del segmento.
       call wspace         ; presentar un espacio, blanco sobre fondo negro.
       call segmov         ; mover el segmento.
       call segcol         ; mirar la colisión de nueva posición del segmento.
       ld a,(ix)           ; verificar si el segmento está activado.
       inc a               ; para la rutina de detección de colisiones.
       ret z               ; está activo, por tanto pasa a muerto.
       call segxy          ; actualizar las coordenadas del segmento.
       ld a,2              ; código de atributo = 2, segmento rojo.
       ld (23695),a        ; actualizar los atributos temporales.
       ld a,146            ; UDG 'C' para presentar el segmento.
       rst 16
       ret
segmov ld a,(ix+1)         ; coordenada x.
       ld c,a              ; área x GP.
       ld a,(ix+2)         ; coordenada y.
       ld b,a              ; área y GP.
       ld a,(ix)           ; indicador de estado.
       and a               ; ¿está el segmento en el borde izquierdo?
       jr z,segml          ; ir a la izquierda, saltar según ese bit de código.

; ¡ahora el segmento se mueve a la derecha!

segmr  ld a,(ix+2)         ; coordenada y.
       cp 31               ; ¿está en el borde derecho de la pantalla?
```

```
       jr z,segmd          ; si, mover el segmento hacia abajo.
       inc a               ; ver la izquierda.
       ld b,a              ; actualizar la coordenada y GP.
       call atadd          ; mirar el atributo de esa dirección.
       cp 68               ; setas son brillo (64) + verde (4).
       jr z,segmd          ; seta a la derecha, moverse hacia abajo.
       inc (ix+2)          ; sin obstáculos, seguir hacia la derecha.
       ret

; ¡ahora el segmento se mueve a la izquierda!

segml  ld a,(ix+2)         ; coordenada y.
       and a               ; ¿está en el borde izquierdo de la pantalla?
       jr z,segmd          ; si, mover el segmento hacia abajo.
       dec a               ; ver la derecha.
       ld b,a              ; actualizar la coordenada y GP.
       call atadd          ; mirar el atributo en la dirección (dispx,dispy).
       cp 68               ; setas son brillo (64) + verde (4).
       jr z,segmd          ; seta a la izquierda, moverse abajo.
       dec (ix+2)          ; sin obstáculos, seguir a la izquierda.
       ret

; ¡ahora el segmento se mueve hacia abajo!

segmd  ld a,(ix)           ; dirección del segmento.
       xor 1               ; cambiarla.
       ld (ix),a           ; guardar la nueva dirección.
       ld a,(ix+1)         ; coordenada y.
       cp 21               ; ¿llegamos al final de la pantalla?
       jr z,segmt          ; si, mover el segmento al inicio.


; En este momento nos estamos moviendo hacia abajo independientemente de las
; setas que pueden bloquear la trayectoria del segmento. Cualquier cosa en el
; camino del segmento será borrada.

       inc (ix+1)          ; no ha llegado a la parte inferior, bajar.
       ret

; mover segmento a la parte superior de la pantalla.

segmt  xor a               ; igual que ld a,0 pero ahorra 1 byte.
       ld (ix+1),a         ; nueva coordenada x = inicio de la pantalla.
       ret

; Detección de colisiones segmento.
; Comprueba la existencia de colisiones con el jugador y las balas del jugador.

segcol ld a,(ply)          ; bala posición y.
       cp (ix+2)           ; ¿es idéntico a la coordenada y del segmento?
       jr nz,bulcol        ; coordenada y diferente, mirar la bala en su lugar.
       ld a,(plx)          ; coordenada x del jugador.
       cp (ix+1)           ; ¿igual a la del segmento?
       jr nz,bulcol        ; coordenada x diferente, mirar la bala en su lugar.

; Así que tenemos una colisión con el jugador.

killpl ld (dead),a         ; Establecer indicador de jugador muerto.
       ret

; Vamos a ver de una colisión con el disparo del jugador.
```

```
bulcol ld a,(pbx)          ; bala posición x.
       inc a               ; ¿tiene el valor por defecto?
       ret z               ; si, no hay disparo que mirar.
       cp (ix+1)           ; ¿es coordenada x del disparo igual al segmento?
       ret nz              ; no, entonces no hay colisión.
       ld a,(pby)          ; bala posición y.
       cp (ix+2)           ; ¿es igual a la y del segmento?
       ret nz              ; no, esta vez no hay colisión.

; Tenemos una colisión con el disparo del jugador.

       call dbull          ; borrar disparo.
       ld a,22             ; código para AT.
       rst 16
       ld a,(pbx)          ; coordenada vertical de la bala del jugador.
       inc a               ; 1 linea abajo.
       rst 16              ; establecer posición vertical de la seta.
       ld a,(pby)          ; posición horizontal de la bala.
       rst 16              ; establecer la coordenada horizontal.
       ld a,16             ; código ASCII para controlar tinta.
       rst 16
       ld a,4              ; 4 = color verde.
       rst 16              ; ¡tenemos todas las setas de ese color!
       ld a,145            ; UDG 'B' es el gráfico de la seta.
       rst 16              ; poner la seta en la pantalla.
       call kilbul         ; matar el proyectil.
       ld (ix),a           ; matar al segmento.
       ld hl,numseg        ; numero de segmentos.
       dec (hl)            ; decrementarlo.
       ret

; Sencillo generador de números seudo-aleatorio.
; Seguir un puntero a través de la ROM (a partir de una semilla),
; retornando el contenido del byte en esa posición.

random ld hl,(seed)        ; puntero
       ld a,h
       and 31              ; mantenerlo en los primeros 8Kb de ROM.
       ld h,a
       ld a,(hl)           ; Obtener el número "aleatorio" de esa ubicación.
       inc hl              ; Incrementar el puntero.
       ld (seed),hl
       ret
seed   defw 0

; Calcular la dirección del atributo de carácter en (dispx, dispy).

atadd  ld a,c              ; coordenada vertical.
       rrca                ; multiplicar por 32.
       rrca                ; desplazar a la derecha con acarreo 3 veces
       rrca                ; mas rápido que desplazar izquierda 5 veces.
       ld e,a
       and 3
       add a,88            ; 88x256=dirección de los atributos.
       ld d,a
       ld a,e
       and 224
       ld e,a
       ld a,b              ; posición horizontal.
       add a,e
```

```
       ld e,a              ; de=dirección de los atributos.
       ld a,(de)           ; devolver atributo en el acumulador.
       ret

plx    defb 0              ; coordenada x del jugador.
ply    defb 0              ; coordenada y del jugador.
pbx    defb 255            ; coordenadas del disparo del jugador.
pby    defb 255
dead   defb 0              ; flag - el jugador muere cuando no sea cero.

; gráficos UDG.

blocks defb 16,16,56,56,124,124,254,254    ; base del jugador.
       defb 24,126,255,255,60,60,60,60     ; seta.
       defb 24,126,126,255,255,126,126,24  ; segmento.
       defb 0,102,102,102,102,102,102,0    ; proyectil del jugador.

; Tabla de segmentos.
; Formato: 3 bytes por entrada, 10 segmentos.
; byte 1: 255=segmento apagado, 0=izquierda, 1=derecha.
; byte 2: coordenada x (vertical).
; byte 3: coordenada y (horizontal).

segmnt defb 0,0,0          ; segmento 1.
       defb 0,0,0          ; segmento 2.
       defb 0,0,0          ; segmento 3.
       defb 0,0,0          ; segmento 4.
       defb 0,0,0          ; segmento 5.
       defb 0,0,0          ; segmento 6.
       defb 0,0,0          ; segmento 7.
       defb 0,0,0          ; segmento 8.
       defb 0,0,0          ; segmento 9.
       defb 0,0,0          ; segmento 10.
```

Descarga el programa desde aquí

***NdT**: Esta es la última versión del juego, a partir de este punto lo que se habla en el resto del documento ya no se aplica a este juego, que se da por finalizado.*

Pero, un momento, ¿por qué hay dos pruebas de detección de colisiones en lugar de una? Bueno, imagina que la base de la pistola del jugador está una celda de carácter a la izquierda de un segmento del ciempiés. El jugador se mueve a la derecha y el segmento se mueve hacia la izquierda. En la siguiente trama el segmento se movería a la celda ocupada por el jugador, y el jugador se movería a la posición ocupada por el segmento en el cuadro anterior, jugador y segmento de ciempiés pasarían directamente uno a través del otro y una comprobación de detección de colisión no puede dejar de tener esto en cuenta. Mediante la comprobación de colisión después de que el jugador se mueve, y luego otra vez después de que los segmentos de ciempiés se han movido, podemos evitar el problema.

Colisiones entre Sprites

Esto es suficiente para nuestro juego, pero la mayoría de juegos de Spectrum utilizan sprites en lugar de UDG, por lo que en el siguiente capítulo veremos cómo se pueden usar los sprites. Para la detección de colisión, el mismo principio de comprobación de coordenadas se puede utilizar para detectar colisiones entre sprites. Restar las coordenadas del primero de los sprites de los del segundo, verificar la diferencia y si está dentro de la gama de tamaños de los dos sprites combinados tenemos una colisión en ese eje. Una prueba de colisión simple para dos sprites de 16x16 pixel podría ser algo como esto:

```
; Verificar (l,h) para su colisión con (c,b), aplicación estricta.

colx16 ld a,l              ; coordenada x.
       sub c               ; restar x.
       add a,15            ; añadir distancia máxima.
       cp 31               ; ¿está en rango de x?
       ret nc              ; no,  están separados.
       ld a,h              ; coordenada y.
       sub b               ; restar y.
       add a,15            ; añadir distancia máxima.
       cp 31               ; ¿está en rango de y?
       ret                 ; poner el indicador de colisión desde aquí.
```

Hay una desventaja con este método. Si los sprites no se rellenan por completo a sus límites de 16x16 píxeles la detección de colisiones parece ser demasiado estricta, por lo que las colisiones se producen cuando los sprites están muy cerca, pero no cuando en realidad se están tocando. Una verificación un poco menos sensible implicaría recortar las esquinas de los sprites en una forma más octogonal, sobre todo si tienen tus sprites esquinas redondeadas. El siguiente proceso funciona mediante la adición a las coordenadas **x** e **y** de las diferencias y la comprobación de que están por debajo de un cierto límite. Para una colisión entre dos sprites de **16 x 16** la coordenada de máxima distancias son 15 píxeles para cada eje, mediante la comprobación de que las diferencias entre **x** e **y** son de 25 o menos estamos efectivamente recortando un triángulo de **5 x 5 x 5** píxeles de cada esquina.

```
; Verificar (l, h) para colisión con (c, b), recortando las esquinas.

colc16 ld a,l              ; coordenada x.
       sub c               ; restar x.
       jr nc,colc1a        ; ¿resultado positivo?
       neg                 ; pasar negativo a positivo.
colc1a cp 16               ; ¿en el rango de x?
       ret nc              ; no, están separados.
       ld e,a              ; guardar la diferencia.

       ld a,h              ; coordenada y.
       sub b               ; restar y.
       jr nc,colc1b        ; ¿resultado positivo?
       neg                 ; pasar negativo a positivo.
colc1b cp 16               ; ¿en el rango de y?
       ret nc              ; no, están separados.
```

```
       add a,e              ; añadir la diferencia en x.
       cp 26                ; ¿solo los 5 pixels de la esquina se tocan?
       ret                  ; actuar cuando hay colisión.
```

Capítulo 8: Sprites

***NdT**: El tema del manejo de la pantalla en modo gráfico en los Spectrum es complejo, el diseño económico de la ULA y la memoria de vídeo hizo que se dividiera la pantalla en tres zonas de 64 líneas, y que en cada zona las líneas estuvieran ordenadas por el barrido de carácter y no consecutivamente, de esta forma guarda las líneas en el orden Zona 1 [1-9-17...57 - 2-10-18...58 - 3-11-19..59 - 8,16, 24...64] Zona 2[65, 73... - 65, 74... - 66, 75...] Zona 3 [129... - 130...], separando la información de color (zona de atributos) en zonas de 8x8 pixels justo después. Las pantallas de carga de los juegos lo hacen en este orden, por lo que se aprecia la distribución de líneas durante su carga. Revise alguna explicación mas detallada para entenderlo, es vital de aquí en adelante.*

Convirtiendo Posiciones de Pixel a Dirección de Pantalla

Los UDG y gráficos de carácter están muy bien, pero los mejores juegos suelen utilizar sprites y no hay rutinas de la ROM para ayudarnos aquí, ya que Sir Clive no diseñó el Spectrum como una máquina de juegos. Tarde o temprano un programador de juegos tiene que afrontar el espinoso tema del torpe diseño de la pantalla del Spectrum. Es un tema difícil convertir las coordenadas **x** e **y** de pixels en direcciones de la pantalla, pero hay un par de métodos que podemos emplear para hacerlo.

Usando una tabla de consulta de direcciones de la pantalla

El primer método es usar una tabla de direcciones pre calculadas, que contiene la dirección de la pantalla para cada una de las 192 líneas del Spectrum, tal como este o una similar:

```
       xor a                ; limpiar bandera de acarreo y el acumulador.
       ld d,a               ; borrar el byte alto de DE.
       ld a,(xcoord)        ; posición x.
       rla                  ; desplazar a la izquierda para multiplicar por 2.
       ld e,a               ; ponerlo en el bit bajo del par DE.
       rl d                 ; desplazar el bit alto al byte alto.
       ld hl,addtab         ; tabla de direcciones de pantalla.
       add hl,de            ; apuntar a la entrada de la tabla.
       ld e,(hl)            ; byte bajo de la dirección de pantalla.
       inc hl               ; apuntar al byte alto.
       ld d,(hl)            ; byte alto de la dirección de pantalla.
       ld a,(ycoord)        ; posición horizontal.
       rra                  ; dividir por dos.
       rra                  ; y luego por cuatro.
       rra                  ; desplazar de nuevo para dividir por ocho.
       and 31               ; enmascarar la distancia hasta los bits desplazados.
       add a,e              ; sumar a la dirección de inicio de la línea.
       ld e,a               ; nuevo valor del registro E.
       ret                  ; volver con la dirección en la pantalla en DE.

.
.
```

```
addtab defw 16384
       defw 16640
       defw 16896
.
.
.
```

En el lado positivo esto es muy rápido, pero significa tener que almacenar cada una de las direcciones de las 192 líneas en una tabla, ocupando 384 bytes que podrían estar mejor empleados en otros usos.

Calculando la Dirección de la Pantalla

El segundo método implica el cálculo de la dirección por nosotros mismos y no requiere una consulta en la tabla de direcciones. Al hacer esto debemos tener en cuenta tres cosas: en que tercio de la pantalla está el punto, que línea de carácter es la mas cercana, y la línea de pixels sobre la que cae esa celda. El uso juicioso del operador **and** nos ayudará a calcular los tres. Es un tema complicado, así que por favor ten paciencia conmigo en mi esfuerzo para explicar cómo funciona (***NdT**: y con el traductor que hace lo que puede*).

Podemos establecer en cuál de los tres segmentos de la pantalla está situado un punto tomando la coordenada vertical y enmascaramiento los seis bits menos significativos para dejar un valor de 0, 64 o 128 para cada uno de los segmentos separados de 64 pixels de alto. Como los bytes altos de las direcciones de los 3 segmentos de la pantalla son 64, 72 y 80, con una diferencia de 8 al pasar de un segmento a otro, se toma este valor enmascarado y se divide entre 8, lo que nos dará un valor de 0, 8 o 16. A continuación, añadir 64 para darnos el byte alto del segmento de pantalla.

Cada segmento se divide en 8 posiciones en celdas de caracteres, lo que son en total 32 bytes separados, por lo que para encontrar ese valor de nuestra dirección tomamos la coordenada vertical, la máscara de distancia de los dos bits más significativos se utiliza para determinar el segmento junto con los tres bits menos significativos que determinan la posición de pixel. La instrucción **and 56** lo harán muy bien. Esto nos da la posición de celda de carácter como un múltiplo de 8, y como las líneas de caracteres son de 32 bytes separados, multiplicamos esto por 4 y colocamos nuestro número en el byte bajo de la dirección de la pantalla.

Por último, las celdas de caracteres se dividen en líneas de pixels de 256 bytes de diferencia, por lo que una vez más tomamos su coordenada vertical, con la máscara de distancia excepto los bits que determinan el uso de la línea usando **and 7**, y añadimos el resultado al byte alto. Eso nos dará nuestra dirección vertical de la pantalla. A partir de ahí tomamos nuestra coordenada horizontal, se divide por 8 y lo añadimos a nuestra dirección.

Aquí presento una rutina que devuelve una dirección de pantalla para las coordenadas (x, y) ubicadas en el par de registro **DE**. Se podría modificar fácilmente para devolver la dirección en los registros **HL** o **BC** si se desea.

```
scadd  ld a,(xcoord)       ; recuperar la coordenada vertical.
       ld e,a              ; guardarla en e.

; Encuentra línea dentro de la celda.

       and 7               ; linea 0-7 en el cuadrado de caracteres.
       add a,64            ; 64 * 256 = 16384 = inicio de la pantalla.
       ld d,a              ; d = linea * 256.

; Encuentra en que tercio de la pantalla estamos.

       ld a,e              ; restaurar la vertical.
       and 192             ; segmento 0, 1 o 2 multiplicamos por 64.
       rrca                ; dividirlo por 8.
       rrca
       rrca                ; segmento 0-2 multiplicado por 8.
       add a,d             ; sumar a+d obtiene dirección de inicio del segmento.
       ld d,a

; Encuentra la celda de carácter dentro del segmento.

       ld a,e              ; 8 casillas de caracteres por segmento.
       rlca                ; dividir x por 8 y multiplicarlo por 32,
       rlca                ; siguiente cálculo: multiplicar por 4.
       and 224             ; enmascarar los bits que no queremos.
       ld e,a              ; cálculo de la coordenada vertical completo.

; Agregar el elemento horizontal.

       ld a,(ycoord)       ; coordenada y.
       rrca                ; sólo es necesario dividir por 8.
       rrca
       rrca
       and 31              ; cuadrados 0 a 31 a través de la pantalla.
       add a,e             ; añadir al total de la medida.
       ld e,a              ; de = dirección de la pantalla.
       ret
```

Moviendo

Una vez que se ha establecido la dirección tenemos que considerar cómo desplazar nuestros gráficos de su posición. Los tres bits bajos de la coordenada horizontal indican cuantos cambios de píxeles se necesitan. Una forma lenta para representar un píxel sería hacer una llamada a la rutina **SCADD** anterior, realizar un **and 7** en la coordenada horizontal, a continuación desplazar a la derecha un píxel de cero a siete veces en función del resultado antes de volcar a la pantalla

Una rutina de desplazamiento de sprites funciona de la misma manera. La imagen gráfica se toma de la memoria una línea a la vez, se desplaza a su posición y luego se coloca en la pantalla antes de pasar a la siguiente línea de abajo y repetir el proceso. Podríamos escribir una rutina de sprites que calcula la dirección de la pantalla para cada línea trazada, y de hecho, la primera rutina de sprites que escribí trabajaba de esa manera. Un método simple consiste en determinar si nos estamos moviendo dentro de una celda de carácter, cruzando los carácter límites de las celdas, o cruzando un límite de segmento con un par de instrucciones **and** para aumentar o disminuir la dirección de la

pantalla en consecuencia. Poner simplemente **and 63** devolverá cero si la nueva posición vertical cruza un segmento, **and 7** devolverá cero si está cruzando un límite de celda de carácter, y cualquier otra cosa significa que una nueva línea se encuentra dentro de la misma celda de carácter que la línea anterior.

Esta es una rutina de movimiento de sprites que hace uso de la rutina **SCADD** anterior. Para usarla, simplemente configurar las coordenadas en **dispx** y **dispy**, apuntar el par de registros **bc** al gráfico del sprite, y **call sprite**:

```
sprit7 xor 7              ; complementa los últimos 3 bits.
       inc a              ; ¡agrega uno por suerte!
sprit3 rl d               ; rotar izquierda...
       rl c               ; ...en el centro del byte...
       rl e               ; ...y a la izquierda de la celda de carácter.
       dec a              ; contar los cambios que hemos hecho.
       jr nz,sprit3       ; regresar hasta que los movimientos estén completos.

; Línea de la imagen de sprite ahora en e+c+d, lo necesitamos en forma c+d+e

       ld a,e             ; borde izquierdo de la imagen está en e.
       ld e,d             ; poner borde derecho en su lugar.
       ld d,c             ; bit central va en d
       ld c,a             ; y el borde izquierdo de nuevo en c.
       jr sprit0          ; hemos hecho el cambio para transferir a la
pantalla.

sprite ld a,(dispx)       ; dibuja el sprite (hl).
       ld (tmp1),a        ; guardar vertical.
       call scadd         ; calcular dirección de la pantalla.
       ld a,16            ; altura del sprite en pixels.
sprit1 ex af,af'          ; guardar el contador de bucles.
       push de            ; guardar dirección de pantalla.
       ld c,(hl)          ; primer gráfico del sprite.
       inc hl             ; incrementar el puntero de datos del sprite.
       ld d,(hl)          ; siguiente bit de la imagen del sprite.
       inc hl             ; apuntar a la siguiente fila de datos del sprite.
       ld (tmp0),hl       ; guardar en tmp0 para más adelante.
       ld e,0             ; byte derecho en blanco por ahora.
       ld a,b             ; b guarda la posición y.
       and 7              ; ¿estamos a caballo entre celdas de caracteres?
       jr z,sprit0        ; no estamos a caballo, no molesta al desplazamiento.
       cp 5               ; ¿necesitamos 5 o más desplazamientos a la derecha?
       jr nc,sprit7       ; sí, desplazar a la izquierda que es más rápido.
       and a              ; Uy, la bandera de acarreo se establece.
sprit2 rr c               ; rotar a la izquierda el byte derecho...
       rr d               ; ...Hasta el byte medio...
       rr e               ; ...la byte derecho.
       dec a              ; un turno menos que hacer.
       jr nz,sprit2       ; repetir hasta que todos los turnos estén completos.
sprit0 pop hl             ; sacar la dirección de la pantalla en la pila.
       ld a,(hl)          ; ya está lista.
       xor c              ; fusionar con los datos de la imagen.
       ld (hl),a          ; colocar en la pantalla.
       inc l              ; siguiente celda de carácter por la derecha.
       ld a,(hl)          ; ya estaba antes.
       xor d              ; fusionarse con el centro de la imagen.
       ld (hl),a          ; poner de nuevo en la pantalla.
```

```
       inc hl              ; siguiente bit del área de la pantalla.
       ld a,(hl)           ; lo que ya está allí.
       xor e               ; borde derecho de los datos de imagen del sprite.
       ld (hl),a           ; poner en pantalla.
       ld a,(tmp1)         ; coordenada vertical temporal.
       inc a               ; siguiente línea de abajo.
       ld (tmp1),a         ; almacenar nueva posición.
       and 63              ; ¿nos movemos al siguiente tercio de pantalla?
       jr z,sprit4         ; sí, encontrar el próximo segmento.
       and 7               ; ¿entrando en celda de carácter siguiente?
       jr z,sprit5         ; Sí, encuentre siguiente fila.
       dec hl              ; izquierda 2 bytes.
       dec l               ; es está el límite a horcajadas de 256 bytes aquí.
       inc h               ; siguiente fila de esta celda de carácter.
sprit6 ex de,hl            ; Dirección de pantalla en de.
       ld hl,(tmp0)        ; restaurar la dirección del gráfico.
       ex af,af'           ; restaurar el contador del bucle.
       dec a               ; decrementarlo.
       jp nz,sprit1        ; no alcanzado el borde inferior del sprite, repetir.
       ret                 ; trabajo hecho.
sprit4 ld de,30            ; el siguiente segmento es de 30 bytes.
       add hl,de           ; añadir a la dirección de la pantalla.
       jp sprit6           ; repetir.
sprit5 ld de,63774         ; menos 1762.
       add hl,de           ; restar 1762 de la dirección física de la pantalla.
       jp sprit6           ; volver al bucle.
```

Como puedes ver, esta rutina utiliza la instrucción **XOR** para combinar el sprite con el fondo de la pantalla, lo que funciona de la misma manera que **PRINT OVER 1** en el sinclair BASIC. El sprite se fusionó con los gráficos que ya están presentes en la pantalla, lo que puede parecer desordenado. Para borrar un sprite solo hay que volver a mostrarlo y la imagen se desvanece mágicamente.

Si quisiéramos dibujar un sprite encima de algo que ya está en la pantalla necesitaríamos algunas rutinas adicionales, aunque similares a la anterior. Una podría usarse para almacenar los gráficos en la pantalla en una memoria intermedia, de modo que la parte de la pantalla podría ser redibujada cuando se elimina el sprite. La siguiente rutina aplicaría una máscara al sprite para eliminar los píxeles de alrededor y de detrás del sprite usando **and** u **or**, a continuación el sprite finalmente podría ponerse encima. Otra rutina podría ser necesaria para restaurar la parte pertinente de la pantalla a su estado anterior cuando se suprime el sprite. Sin embargo, esto llevaría mucho tiempo de CPU para conseguirlo, así que mi consejo sería no molestase a no ser que el juego use algo llamado doble buffer (también conocida como la técnica de la pantalla trasera), o si está utilizando un sprite pre-desplazado, lo que discutiremos en breve.

Otro método que puedes considerar es la posibilidad de hacer sprites que parecen pasar por detrás de los objetos del fondo, un truco que puedes haber visto en "Haunted House" o en "Egghead in Space". Si bien este método es útil para reducir el choque de colores, requiere usar una parte considerable de la memoria. En ambos juegos una pantalla de máscara ficticia de 6Kb se encuentra en la dirección 24576, y cada byte de datos del sprite se añade a los datos en la pantalla oculta antes de forzar su salida a la pantalla física ubicada en la dirección 16384. Debido a que la pantalla física y la pantalla de máscara ficticia eran exactamente de 8K era posible intercambiarlas entre ellas con

alternar el bit 5 del registro h. Para hacer esto en la rutina de sprites anterior nuestro **sprit0** podría tener este aspecto:

```
sprit0 pop hl               ; sacar la dirección de la pantalla en la pila.
       set 5,h              ; dirección de la pantalla replicada.
       ld a,(hl)            ; ya está lista.
       and c                ; fusionar con los datos de la imagen.
       res 5,h              ; dirección de la pantalla física.
       xor (hl)             ; mezclar en los datos de la imagen.
       ld (hl),a            ; colocar en la pantalla.
       inc l                ; siguiente celda de carácter por la derecha.
       set 5,h              ; dirección de la pantalla replicada.
       ld a,(hl)            ; ya estaba antes.
       and d                ; encarcara con el bit centrsl de la imagen.
       res 5,h              ; dirección de la pantalla física.
       xor (hl)             ; mezclar con los datos de la imagen.
       ld (hl),a            ; fusionarse con el centro de la imagen.
       - eliminado -
       inc hl               ; siguiente bit del área de la pantalla.
       set 5,h              ; dirección de la pantalla replicada.
       ld a,(hl)            ; lo que ya está allí.
       and e                ; enmascara borde derecho de datos imagen del sprite.
       res 5,h              ; dirección de la pantalla física.
       xor (hl)             ; mezclar con los datos de la imagen.
       ld (hl),a            ; poner en pantalla.
       ld a,(tmp1)          ; coordenada vertical temporal.
```

Este es otro método para calcular la dirección de la siguiente línea inferior. Como antes, lo calculamos desde nuestra dirección actual. Hacerlo así siempre es bastante complicado por la forma en que la pantalla está estructurada. La mayoría de las veces nuestra siguiente línea está a 256 bytes de la anterior, pero cada vez que se cruce la frontera de celda de carácter tenemos que restar 1760, y cuando se mueve a un tercio de la pantalla a la siguiente tenemos que añadir 32. Este pequeño fragmento calcula la dirección de la siguiente línea en el par de registro **HL** sin alterar **DE**.

```
; Línea dibujada, ahora buscamos la siguiente dirección objetivo.

       inc h                ; incrementar pixel.
       ld a,h               ; obtener la dirección del pixel.
       and 7                ; ¿posición de carácter a caballo?
       ret nz               ; no, estamos ya en la línea siguiente.
       ld a,h               ; obtener la dirección del pixel.
       sub 8                ; restar 8 para el inicio del segmento.
       ld h,a               ; nuevo byte alto de la dirección.
       ld a,l               ; obtener byte bajo de dirección.
       add a,32             ; una linea abajo.
       ld l,a               ; nuevo byte bajo.
       ret nc               ; no se ha alcanzado todavía siguiente segmento.
       ld a,h               ; dirección alta.
       add a,8              ; sumar 8 al siguiente segmento.
       ld h,a               ; nuevo byte alto.
```

Sprites Pre-desplazados

Una rutina de movimiento para spriteS tiene una gran desventaja: su falta de velocidad. Cambiar todos esos datos gráficos de posición de toma tiempo, y si tu juego necesita muchos sprites que rebotan por la pantalla, debes considerar el uso de sprites pre-desplazados en su lugar. Esto requiere hasta ocho copias separadas de la imagen del sprite, una por cada una de las posiciones de pixel desplazados (***NdT****: Se refiere a mover el sprite a través de los recuadros de caracter de 8x8 del Spectrum*). Por lo general los sprites se mueven en saltos de 2 pixels y así se usan solo 4 copias de cada sprite. Luego es simplemente cuestión de calcular qué imagen del sprite utilizar en función de la alineación horizontal del sprite, calcular la dirección de la pantalla, y copiar la imagen del sprite a la pantalla. Si bien este método es mucho más rápido, es muy caro en términos de memoria. Una rutina de pre-desplazamiento de sprites requiere 32 bytes para presentar un sprite de 16x16 pixels, un sprite pre-desplazado con 4 posiciones requiere 128 bytes para la misma imagen, ¡y si es de 8 posiciones la friolera de 256! Escribir un juego de Spectrum es siempre un compromiso entre velocidad y memoria disponible.

Puede que no quieras necesariamente la misma imagen del sprite en cada posición pre-desplazada. Por ejemplo, cambiando la posición de las piernas de un sprite en cada uno de las posiciones pre-desplazadas del sprite puedes animarlo para parecer como si caminara de izquierda a derecha mientras se mueve por la pantalla. Recuerda que deben coincidir con las piernas del personaje el número de pixels que se mueve cada trama. Si va a mover un sprite 2 pixels cada trama, es importante hacer que los miembros se muevan 2 pixels entre cuadros. Menos de que esto hará que el sprite parezca como patinando sobre hielo, si es mas parecerá estar luchando para lograr agarrarse. Te voy a contar un pequeño secreto: lo creas o no esto puede afectar en realidad a la forma en que el juego se percibe, así que conseguir la animación adecuada es importante.

El método de los Bytes de exploración

Si estás mostrando sprites directamente en la pantalla y deseas mostrar más de un par, es posible que encuentres un problema. Borrar, mover y volver a mostrar necesita tiempo para completarse, y mientras esto sucede la línea de exploración del televisión está en movimiento. Si la línea de exploración alcanza al sprite mientras se visualiza o se elimina, se puede experimentar un parpadeo. Una forma de evitar esto es utilizar el método del byte de exploración o una variación del mismo.

El método del byte de exploración consiste en dibujar el nuevo sprite mientras se borra al mismo tiempo el antíguo, byte a byte, o línea por línea. Si la línea de exploración alcanza a tu sprite, el único pequeño parpadeo que se ve es el del byte o línea en que se está en ese momento. Por tanto en realidad no importa cuántos sprites tienes en la pantalla o dónde están, porque nunca parpadean. El lado negativo es que esto puede ser difícil de poner en práctica, ya que consiste en mantener dos copias de cada coordenada de los sprites, la imagen, el marco y toda la información relevante, y requiere una gran cantidad de registros. Afortunadamente, el Z80A tiene una instrucción muy útil para intercambiar entre los dos bancos de registros muy rápidamente: **EXX**.

Una vez que hemos llamado a las rutinas para elaborar los datos gráficos de tu antiguo sprite en **de**, con la dirección de la pantalla en **hl** y tal vez una máscara en **bc**, usaríamos entonces la instrucción **exx** para intercambiarlos con el conjunto de registros alternativos, para a continuación llamar a las rutinas de nuevo para poner el nuevo puntero de datos del sprite, la dirección y la máscara de pantalla en los registros **hl** y **bc**. A continuación, se trata de dibujar cada línea a la vez, cálcular la dirección de la siguiente línea de la pantalla, cambiar los registros con **exx**, trazar la línea del nuevo sprite, cálculo de la siguiente dirección de pantalla, y volver a intercambiar los registros de nuevo. Se repite en un bucle este proceso y va a reemplazar un sprite sin tener que eliminarlo completamente de la pantalla.

No olvides que también debes disponer de una rutina sencilla de manejo de sprites que presente un sprite sin borrar el antiguo, o que borre un sprite sin que aparezca de nuevo. De lo contrario, no serás capaz de poner nuevos sprites en la pantalla o eliminarlos cuando ya no son necesarios.

Capítulo 9: Gráficos de fondo

Presentando bloques

Digamos que queremos escribir un sencillo juego de laberinto por pantalla. Tenemos que mostrar las paredes alrededor de las cuales el sprite del jugador debe ser movido, y la mejor manera de hacer esto es crear una tabla de bloques que se transfieren a la pantalla secuencialmente. A medida que avanzamos a través de la tabla encontramos la dirección del bloque gráfico, cálculamos su dirección de pantalla y volcamos el carácter a la pantalla.

Vamos a empezar con la rutina de visualización de caracteres. A diferencia de la rutina que necesitamos para que el sprite se acople a posiciones de caracteres, por suerte es más fácil calcular una dirección de pantalla para una posición de carácter de lo que lo es para píxeles individuales.

Hay 24 posiciones de celdas de caracter verticales y 32 posiciones horizontales en la pantalla del spectrum, por lo que nuestras coordenadas estarán entre (0,0) y (23,31). Las filas 0-7 caen en el primer segmento de pantalla, 8-15 en la sección media de la pantalla, y las posiciones 16-23 en la tercera parte de la pantalla. Tenemos suerte, el byte alto de la dirección de pantalla para cada segmento se incrementa en 8 de un segmento a otro, por lo que tomando el número de celdas verticales y realizando un **and 24** inmediatamente obtenemos el desplazamiento del comienzo del segmento de pantalla que debemos abordar allí mismo. Añadir 64 para el inicio de la pantalla del Spectrum y tenemos el byte alto de nuestra dirección. Entonces tenemos que encontrar la celda de carácter correcta dentro de cada segmento, por lo que tomamos de nuevo la coordenada vertical, y esta vez usamos **and 7** para determinar cuál de las siete filas estamos tratando de encontrar. Multiplicamos esto por el ancho en caracteres de la pantalla (32) y añadimos el número de celda horizontal para encontrar el byte bajo de la dirección de la pantalla. Un ejemplo adecuado es el siguiente:

```
; retorna dirección de la celda de carácter en (b, c).

chadd  ld a,b              ; posición vertical.
       and 24              ; ¿qué segmento, 0, 1 o 2?
       add a,64            ; 64*256 = 16384, memoria de pantalla del Spectrum.
       ld d,a              ; este es nuestro byte alto.
       ld a,b              ; ¿cuan es la posición vertical ahora?
       and 7               ; ¿qué fila dentro del segmento?
       rrca                ; multiplcar fila por 32.
       rrca
       rrca
       ld e,a              ; byte bajo.
       ld a,c              ; añadir la coordenada y.
       add a,e             ; mezclar con el byte bajo.
       ld e,a              ; dirección de pantalla en de.
```

Una vez que tenemos nuestra dirección de pantalla es un proceso sencillo el volcar el carácter a pantalla. Mientras que no estamos cruzando los límites de las celdas de caracteres la siguiente línea

de la pantalla siempre caerá 256 bytes después de la de su predecesor, por lo que hay que incrementar el byte alto de dirección para encontrar la siguiente línea.

```
; Mostrar el caracter hl en (b, c).

char   call chadd          ; encontrar la dirección en pantalla del caracter.
       ld b,8              ; numero del pixel alto.
char0  ld a,(hl)           ; fuente del grafico.
       ld (de),a           ; transferir a la pantalla.
       inc hl              ; siguiente trozo de datos.
       inc d               ; siguiente linea de pixels.
       djnz char0          ; repetir
       ret
```

En cuanto a la coloración de nuestro bloque, lo hemos cubierto en el capítulo sobre detección de colisiones de atributos sencilla. La rutina **atadd** nos dará la dirección de un atributo de celda de carácter en (b, c).

Por último, tenemos que decidir qué bloque mostrar en cada celda de pantalla. Podemos pensar que tenemos 3 tipos de bloques para nuestro juego, podríamos usar un bloque de tipo 0 para un espacio, tipo 1 para el muro y tipo 2 para una puerta. Arreglaríamos los gráficos y atributos para cada bloque en tablas separadas en el mismo orden:

```
blocks equ $

; block 0 = caracter para espacio.

       defb 0,0,0,0,0,0,0,0

; block 1 = muro.

       defb 1,1,1,255,16,16,16,255

; block 2 = puerta.

       defb 6,9,9,14,16,32,80,32

attrs  equ $

; block 0 = espacio.

       defb 71

; block 1 = muro.

       defb 22

; block 2 = puerta.

       defb 70
```

A medida que avanzamos a través de nuestro tablero de hasta 24 filas y 32 columnas de bloques de laberinto cargamos el número del bloque en el acumulador, y llamamos a las rutinas **fblock** y **fattr** a continuación para obtener las direcciones del gráfico de origen y su atributo.

```
; Encontrar celda del gráfico.

fblock rlca               ; multiplicar el número de bloque por 8.
       rlca
       rlca
       ld e,a             ; despalzar a la dirección del grafico.
       ld d,0             ; sin byte alto.
       ld hl,blocks       ; dirección del bloque de caracteres.
       add hl,de          ; apuntar al bloque.
       ret

; Encontrar celda de atributo.

fattr  ld e,a             ; desplazamiento a la dirección del atributo.
       ld d,0             ; sin byte alto.
       ld hl,attrs        ; dirección del bloque de atributos.
       add hl,de          ; apuntar al bloque.
       ret
```

Usar este método significa que nuestros datos del laberinto requiere un byte de RAM por cada celda de carácter. Para un área de juego de 32 celdas de ancho y 16 bloques de alto esto significaría que cada pantalla ocupa 512 bytes de memoria. Eso estaría bien para 20 pantallas de plataformas como en el **Manic Miner**, pero si quieres un centenar o más de pantallas deberías considerar el uso de bloques más grandes de modo que requiera menos bloques cada pantalla. Mediante el uso bloques de celdas de caracteres de 16 x 16 píxeles en lugar de los 8 x 8 en nuestro ejemplo, cada tabla de pantalla requeriría sólo 128 bytes, lo que significa que podremos exprimir mas la memoria del Spectrum.

Capítulo 10: Marcadores y puntuación más alta

Más rutinas de marcadores

Hasta ahora hemos usado una rutina de puntuación poco sofisticada. Nuestro marcador se guarda como un número de 16 bits almacenado en un par de registros, y para que aparezca hemos hecho uso de la rutina de impresión por pantalla del número de línea de la ROM Sinclair. Ahí están los dos principales inconvenientes de este método, en primer lugar que se limita a números de 0 al 9999, y en segundo lugar que se ve horrible.

Podríamos convertir un número de 16 bits a ASCII por nosotros mismos de esta manera:

```
; Mostrar número pasado en hl, justificado a la derecha.

shwnum ld a,48             ; ceros a la izquierda (usa 32 si prefieres
espacios).
       ld de,10000         ; columna de diez miles.
       call shwdg          ; presentar el dígito.
       ld de,1000          ; columna de miles.
       call shwdg          ; presentar el dígito.
       ld de,100           ; columna de cientos.
       call shwdg          ; presentar el dígito.
       ld de,10            ; columna de decenas.
       call shwdg          ; presentar el dígito.
       or 16               ; el último dígito siempre se presenta.
       ld de,1             ; columna de unidades.
shwdg  and 48              ; borrar acarreo, borrar dígito.
shwdg1 sbc hl,de           ; restar de la columna.
       jr c,shwdg0         ; nada que mostrar.
       or 16               ; algo que mostrar, lo convierto en un dígito.
       inc a               ; incrementar digito.
       jr shwdg1           ; repetir hasta que la columna sea cero.
shwdg0 add hl,de           ; restaurar total.
       push af
       rst 16              ; presentar caracter.
       pop af
       ret
```

Este método funciona bien, aunque todavía estamos limitados a una puntuación de cinco dígitos que no supere 65535. Por un tema de apariencia más profesional completo con ceros a la izquierda, lo que necesitamos para mantener la puntuación como una cadena de dígitos ASCII.

He utilizado la misma técnica de puntuación por algo así como 15 años, no es terriblemente sofisticada pero es lo suficientemente buena para hacer lo que necesitamos. Este método utiliza un carácter ASCII por dígito, lo que hace que sea fácil de visualizar. De paso, esta rutina la tomo de mi juego estilo *Shoot 'em up* **More Tea, Vicar?** (***NdT****: Para llamarse ¿Mas te, párroco? es un arcade de disparos a naves espaciales con scrool horizontal*).

```
score   defb '000000'
uscor   ld a,(hl)           ; valor actual de los dígitos.
        add a,b             ; añadir puntero a este dígito.
        ld (hl),a           ; lugar del nuevo dígito en la cadena.
        cp 58               ; ¿mayor que el valor ASCII para nueve?
        ret c               ; no, tranquilidad.
        sub 10              ; restar 10.
        ld (hl),a           ; meter el nuevo carácter en la cadena.
uscor0  dec hl              ; anterior carácter en la cadena.
        inc (hl)            ; incrementar en uno.
        ld a,(hl)           ; ¿cuál es el nuevo valor?
        cp 58               ; ¿se pasa del ASCII para nueve?
        ret c               ; no, puntuación efectuada.
        sub 10              ; restarle diez.
        ld (hl),a           ; ponerlo de nuevo.
        jp uscor0           ; volver al bucle.
```

Para utilizar esto apunto en **hl** al dígito que nos gustaría aumentar, coloco la cantidad que queremos añadir en el registro **b**, luego llamo a **uscor**. Por ejemplo, para agregar 250 a la puntuación se requieren 6 líneas:

```
; Añadir 250 a la puntuación.

        ld hl,score+3       ; apunto a la columna de las centenas.
        ld b,2              ; 2 cienes = 200.
        call uscor          ; incrementar la puntuación.
        ld hl,score+4       ; apunto a la columna de las decenas.
        ld b,5              ; 5 dieces = 50.
        call uscor          ; incrementar la puntuación.
```

Simple, pero hace el trabajo. Los pedantes no dudarían en señalar que podría hacer utilizando BCD, y que los códigos de operación para esto se encuentran en el conjunto de instrucciones del Z80.

Tablas de puntuación máxima

Las rutinas de puntuación máxima no son especialmente fáciles de escribir para un principiante, pero una vez que has escrito una puede ser reutilizada una y otra vez. El principio básico es que empezamos en la parte inferior de la tabla y seguimos nuestro camino hacia arriba hasta que encontramos una puntuación que es **mayor o igual** a la puntuación del jugador. Entonces desplazamos todos los datos de la tabla hacia abajo a partir de ese punto una posición, y copiamos el nombre del jugador y la puntuación en ese punto de la tabla.

Podemos apuntar con los registros **hl** o **ix** al primer dígito de la puntuación inferior en la tabla y seguir nuestro camino con la comparación de cada dígito con el correspondiente en la puntuación del jugador. Si el dígito en la puntuación del jugador es mayor nos movemos hacia arriba una posición, si es inferior nos detenemos allí y copiamos la puntuación del jugador en la tabla un lugar

hacia arriba. Si los dígitos son los mismos nos movemos al siguiente dígito y repetimos la comprobación hasta que los dígitos son diferentes o hemos comprobado todos los dígitos en la puntuación. Si los resultados son idénticos colocamos la entrada del jugador en el lugar de la tabla que aparece a continuación. Esto se repite hasta que una puntuación en la tabla es mayor que la puntuación del jugador, o se llega a la parte superior de la tabla.

En la creación inicial de la tabla de puntuación puede ser tentador colocar tu propio nombre en la parte superior con una puntuación que sea muy difícil de superar. Trata de resistir esta tentación. Las tablas de puntuación más alta son para que el jugador pueda juzgar su propio desempeño, y no hay necesidad de frustrar al jugador haciendo muy difícil llegar a la primera posición.

Capítulo 11: Movimientos de enemigos

Ya tenemos nuestro fondo cargado y permitimos al jugador mover un sprite a su alrededor, lo que ahora necesitamos son algunos sprites de enemigos que el jugador deba evitar. Un programador novato podría pelear mucho con esto, pero en realidad es mucho más simple de lo que parece.

Enemigos que patrullan

El tipo de enemigo más fácil de programar es el que usa un algoritmo fijo a seguir, o patrulla una ruta predeterminada. Hemos cubierto una de estas técnicas en el juego del ciempiés anterior. Otro ejemplo muy sencillo es la que se encuentra en juegos tales como **Jet Set Willy**, donde un sprite se desplaza en una sola dirección hasta que llega al final de su zona de patrulla, a continuación cambia de dirección y se dirige de nuevo a su punto de partida, antes de cambiar de dirección otra vez e iniciar el ciclo de nuevo. Como se puede imaginar, estas rutinas son increíblemente fáciles de escribir.

En primer lugar hemos de crear en nuestra tabla con la estructura del alien una coordenada de posición mínima y máxima, y la dirección actual. Es generalmente buena idea comentar estas tablas, así que vamos a hacerlo.

```
; tabla de datos del Alien, 6 bytes por alien.
; ix     = tipo de gráfico, como pollo/medusa etc.
; ix + 1 = dirección, 0=arriba, 1=derecha, 2=abajo, 3=izquierda.
; ix + 2 = coordenada x actual.
; ix + 3 = coordenada y actual.
; ix + 4 = mínima coordenada x o y , dependiendo de la dirección.
; ix + 5 = máxima coordenada x o y , dependiendo de la dirección.

altab  defb 0,0,0,0,0,0
       defb 0,0,0,0,0,0
       defb 0,0,0,0,0,0
```

luego para manejar su sprite podríamos escribir algo como esto

```
       ld a,(ix+1)         ; dirección del movimiento del alien.
       rra                 ; rotar bit bajo con acarreo.
       jr nc,movav         ; si no hay acarreo = 0 o 2, debe ser vertical.

; dirección es 1 o 3 por lo que es horizontal.

       rra                 ; rotar siguiente bit con para probar.
       jr nc,movar         ; dirección 1 = mover a la derecha el alien.

; Mover alien a la izquierda.

moval  ld a,(ix+3)         ; obtener la coordenada y.
       sub 2               ; mover hacia la izquierda.
       ld (ix+3),a
       cp (ix+4)           ; ¿se ha llegado al mínimo?
```

```
       jr z,movax          ; sí, cambio de dirección.
       jr c,movax          ; Hay, nos hemos pasado.
       ret

; Mover alien a la derecha.

movar  ld a,(ix+3)         ; obtener la coordenada y.
       add a,2             ; mover hacia la derecha.
       ld (ix+3),a
       cp (ix+5)           ; ¿se ha llegado al máximo?
       jr nc,movax         ; sí, cambio de dirección.
       ret

; Mover alien verticalmente.

movav  rra                 ; probar la dirección.
       jr c,movad          ; dirección 2 es abajo.

; Mover alien hacia arriba.

movau  ld a,(ix+2)         ; obtener la coordenada x.
       sub 2               ; mover hacia arriba.
       ld (ix+2),a
       cp (ix+4)           ; ¿se ha llegado al mínimo?
       jr z,movax          ; sí, cambio de dirección.
       ret

; Mover alien hacia abajo.

movad  ld a,(ix+2)         ; obtener la coordenada x.
       add a,2             ; mover hacia abajo.
       ld (ix+2),a         ; nueva coordenada.
       cp (ix+5)           ; ¿se ha llegado al máximo?
       jr nc,movax         ; sí, cambio de dirección.
       ret

; Cambiar dirección del alien.

movax  ld a,(ix+1)         ; indicador de dirección.
       xor 2               ; cambiar dirección, ya sea
                           ; horizontal o verticalmente.
       ld (ix+1),a         ; establecer una nueva dirección.
       ret
```

Si quisiéramos ir más lejos podríamos introducir una bandera extra en nuestra tabla, ix + 6, para controlar la velocidad del sprite, y sólo moverlo, por ejemplo, cada dos cuadros si se establece el indicador. Aunque es simple de escribir y con bajo uso de memoria, este tipo de movimiento es bastante básico, predecible y de uso limitado. Para enemigos con patrullaje más complicado, por ejemplo, las olas de ataque de alienígenas en un shoot-em-up, necesitamos tablas de coordenadas, y mientras que el código es también fácil de escribir, coordinan las tablas rápidamente se come la memoria, sobre todo si se almacenan ambas coordenadas **x** e **y**. Para acceder a una tabla de este tipo se necesitan dos bytes por sprites que actúan como un puntero a la tabla de coordenadas.

Una sección típica de código se vería así:

```
                        ; NdT: little endian = primero byte bajo
       ld l,(ix+2)      ; puntero de byte bajo, little endian.
       ld h,(ix+3)      ; puntero byte alto.
       ld c,(hl)        ; poner coordenada x en c.
       inc hl           ; punto a la coordenada y.
       ld b,(hl)        ; poner la coordenada y en b.
       inc hl           ; apuntar a la siguiente posición.
       ld (ix+2),l      ; siguiente puntero al byte bajo.
       ld (ix+3),h      ; siguiente puntero al byte alto.
```

A continuación un ejemplo un poco más complicado, muestra una oleada de 8 naves de ataque usando una tabla de coordenadas verticales. La posición horizontal de cada sprite se mueve a la izquierda a una velocidad constante de 2 píxeles por imagen, así que no hay necesidad de preocuparse de guardarla. Se utiliza la rutina de sprites precargados del capítulo 8, de manera que los sprites son un poco "peliculeros", pero eso no es importante ahora.

```
mloop  halt             ; esperar al haz de TV.
       ld ix,entab      ; apuntar a las naves impares.
       call mship       ; mover las naves.
       halt
       ld ix,entab+4    ; apuntar a las naves pares.
       call mship       ; mover las naves de nuevo.
       call gwave       ; generar ondas frescas.
       jp mloop         ; volver al inicio del bucle.

; Mover las naves enemigas.

mship  ld b,4           ; Número a procesar.
mship0 push bc          ; guardar contador.
       ld a,(ix)        ; obtener puntero bajo.
       ld l,a           ; ponerlo en l.
       ld h,(ix+1)      ; obtener byte alto.
       or h             ; comprobar que el puntero está configurado.
       and a            ; ¿lo está?
       call nz,mship1   ; sí, procesarlo.
       ld de,8          ; saltar a la siguiente solo-pero-una entrada.
       add ix,de        ; apuntar al siguiente enemigo.
       pop bc           ; restaurar contador.
       djnz mship0      ; repetir para todos los enemigos.
       ret

mship1 push hl          ; guardar puntero a la coordenada.
       call dship       ; Eliminar esta nave.
       pop hl           ; restaurar coordenadas.
       ld a,(hl)        ; recuperar la siguiente coordenada.
       inc hl           ; mover el puntero allí.
       ld (ix),l        ; nuevo puntero al byte bajo.
       ld (ix+1),h      ; puntero al byte alto.
       ld (ix+2),a      ; establecer coordenada x.
       ld a,(ix+3)      ; recuperar la posición horizontal.
       sub 2            ; mover hacia la izquierda 2 píxeles.
       ld (ix+3),a      ; establecer nueva posición.
       cp 240           ; ¿alcanzamos el borde de la pantalla?
       jp c,dship       ; no por el momento, mostrar en la nueva posición.
       xor a            ; poner a cero el acumulador.
       ld (ix),a        ; borrar el byte bajo del puntero.
       ld (ix+1),a      ; limpiar el byte alto del puntero.
```

```
       ld hl,numenm         ; número de enemigos en pantalla.
       dec (hl)             ; uno menos a los que hacer frente.
       ret

gwave  ld hl,shipc          ; contador de naves.
       dec (hl)             ; una menos.
       ld a,(hl)            ; comprobar nuevo valor.
       cp 128               ; ¿esperando el siguiente ataque?
       jr z,gwave2          ; ataque es inminente por lo que hay que
configurarlo.
       ret nc               ; si.
       and 7                ; ¿es hora de generar una nueva nave?
       ret nz               ; aún no lo es
       ld ix,entab          ; tabla de enemigos.
       ld de,4              ; tamaño de cada entrada.
       ld b,8               ; Número a comprobar.
gwave0 ld a,(ix)            ; byte bajo del puntero.
       ld h,(ix+1)          ; byte alto.
       or h                 ; ¿están a cero?
       jr z,gwave1          ; sí, esta entrada está vacía.
       add ix,de            ; apuntar a la siguiente nave.
       djnz gwave0          ; repetir hasta que encontremos una.
       ret
gwave2 ld hl,wavnum         ; presentar el numero de la oleada.
       ld a,(hl)            ; recuperar la configuración actual.
       inc a                ; siguiente a lo largo.
       and 3                ; empezar de nuevo después de la 4ª oleada.
       ld (hl),a            ; escribir nuevo ajuste.
       ret
gwave1 ld hl,numenm         ; número de enemigos en pantalla.
       inc (hl)             ; uno más con que pelear.
       ld a,(wavnum)        ; numero de oleada.
       ld hl,wavlst         ; punteros de datos de oleadas.
       rlca                 ; multiplico por 2.
       rlca                 ; multiplico por 4.
       ld e,a               ; desplazamiento en e.
       ld d,0               ; sin byte alto.
       add hl,de            ; encontrar la dirección de la oleada.
       ld a,(shipc)         ; contador de naves.
       and 8                ; ¿ataque par o impar?
       rrca                 ; hacerlo múltiplo de 2 en consecuencia.
       rrca
       ld e,a               ; desplazamiento en e.
       ld d,0               ; sin byte alto.
       add hl,de            ; apunto a la primera o a la segunda mitad del
ataque.
       ld e,(hl)            ; byte bajo del puntero del ataque.
       inc hl               ; segundo byte.
       ld d,(hl)            ; byte alto del puntero del ataque.
       ld (ix),e            ; byte bajo del puntero.
       ld (ix+1),d          ; byte alto.
       ld a,(de)            ; Recuperar la primera coordenada.
       ld (ix+2),a          ; establecer x.
       ld (ix+3),240        ; comenzar en el borde derecho de la pantalla.

; Visualización de las naves enemigas.

dship  ld hl,shipg          ; Dirección de los sprites.
       ld b,(ix+3)          ; coordenada y.
       ld c,(ix+2)          ; coordenada x.
       ld (xcoord),bc       ; establecer coordenadas de la rutina de sprites.
```

```
       jp sprite           ; llamar a la rutina de sprites.
shipc  defb 128            ; contador de naves.
numenm defb 0              ; número de enemigos.


; Coordenadas de las oleadas de ataque.
; Sólo la coordenada vertical se almacena ya que todas las naves
; se mueven hacia la izquierda 2 píxeles en cada fotograma.

coord0 defb 40,40,40,40,40,40,40,40
       defb 40,40,40,40,40,40,40,40
       defb 42,44,46,48,50,52,54,56
       defb 58,60,62,64,66,68,70,72
       defb 72,72,72,72,72,72,72,72
       defb 72,72,72,72,72,72,72,72
       defb 70,68,66,64,62,60,58,56
       defb 54,52,50,48,46,44,42,40
       defb 40,40,40,40,40,40,40,40
       defb 40,40,40,40,40,40,40,40
       defb 38,36,34,32,30,28,26,24
       defb 22,20,18,16,14,12,10,8
       defb 6,4,2,0,2,4,6,8
       defb 10,12,14,16,18,20,22,24
       defb 26,28,30,32,34,36,38,40
coord1 defb 136,136,136,136,136,136,136,136
       defb 136,136,136,136,136,136,136,136
       defb 134,132,130,128,126,124,122,120
       defb 118,116,114,112,110,108,106,104
       defb 104,104,104,104,104,104,104,104
       defb 104,104,104,104,104,104,104,104
       defb 106,108,110,112,114,116,118,120
       defb 122,124,126,128,130,132,134,136
       defb 136,136,136,136,136,136,136,136
       defb 136,136,136,136,136,136,136,136
       defb 138,140,142,144,146,148,150,152
       defb 154,156,158,160,162,164,166,168
       defb 170,172,174,176,174,172,170,168
       defb 166,164,162,160,158,156,154,152
       defb 150,148,146,144,142,140,138,136


; Lista de oleadas de ataque.

wavlst defw coord0,coord0,coord1,coord1
       defw coord1,coord0,coord0,coord1

wavnum defb 0              ; puntero actual a la oleada wave pointer.


; Sprite de la nave.

shipg  defb 248,252,48,24,24,48,12,96,24,48,31,243,127,247,255,247
       defb 255,247,127,247,31,243,24,48,12,96,24,48,48,24,248,252

sprit7 xor 7               ; complementa los últimos 3 bits.
       inc a               ; ¡agrega uno por suerte!
sprit3 rl d                ; rotar izquierda...
       rl c                ; ...en el centro del byte...
       rl e                ; ...y a la izquierda de la celda de carácter.
       dec a               ; contar los cambios que hemos hecho.
       jr nz,sprit3        ; regresar hasta que los movimientos estén completos.
```

```
; Línea de la imagen de sprite ahora en e+c+d, lo necesitamos en forma c+d+e

       ld a,e              ; borde izquierdo de la imagen está en e.
       ld e,d              ; poner borde derecho en su lugar.
       ld d,c              ; bit central va en d
       ld c,a              ; y el borde izquierdo de nuevo en c.
       jr sprit0           ; hemos hecho el cambio para transferir a la
pantalla.

sprite ld a,(xcoord)        ; dibuja el sprite (hl).
       ld (tmp1),a         ; guardar vertical.
       call scadd          ; calcular dirección de la pantalla.
       ld a,16             ; altura del sprite en pixeles.
sprit1 ex af,af'           ; guardar el contador de bucles.
       push de             ; guardar dirección de pantalla.
       ld c,(hl)           ; primer gráfico del sprite.
       inc hl              ; incrementar el puntero de datos del sprite.
       ld d,(hl)           ; siguiente bit de la imagen del sprite.
       inc hl              ; apuntar a la siguiente fila de datos del sprite.
       ld (tmp0),hl        ; guardar en tmp0 para más adelante.
       ld e,0              ; byte derecho en blanco por ahora.
       ld a,b              ; b guarda la posición y.
       and 7               ; ¿estamos a caballo entre celdas de caracteres?
       jr z,sprit0         ; no estamos a caballo, no molesta al desplazamiento.
       cp 5                ; ¿necesitamos 5 o más desplazamientos a la derecha?
       jr nc,sprit7        ; sí, desplazar a la izquierda que es más rápido.
       and a               ; hay, la bandera de acarreo se establece.
sprit2 rr c                ; rotar a la izquierda el byte derecho...
       rr d                ; ...Hasta el byte medio...
       rr e                ; ...la byte derecho.
       dec a               ; un turno menos que hacer.
       jr nz,sprit2        ; repetir hasta que todos los turnos estén completos.
sprit0 pop hl              ; sacar la dirección de la pantalla en la pila.
       ld a,(hl)           ; ya está lista.
       xor c               ; fusionar con los datos de la imagen.
       ld (hl),a           ; colocar en la pantalla.
       inc l               ; siguiente celda de carácter por la derecha.
       ld a,(hl)           ; ya estaba antes.
       xor d               ; fusionarse con el centro de la imagen.
       ld (hl),a           ; poner de nuevo en la pantalla.
       inc hl              ; siguiente bit del área de la pantalla.
       ld a,(hl)           ; lo que ya está allí.
       xor e               ; borde derecho de los datos de imagen del sprite.
       ld (hl),a           ; poner en pantalla.
       ld a,(tmp1)         ; coordenada vertical temporal.
       inc a               ; siguiente línea de abajo.
       ld (tmp1),a         ; almacenar nueva posición.
       and 63              ; ¿nos movemos al siguiente tercio de pantalla?
       jr z,sprit4         ; sí, encontrar el próximo segmento.
       and 7               ; ¿entrando en celda de carácter siguiente?
       jr z,sprit5         ; Sí, encuentre siguiente fila.
       dec hl              ; izquierda 2 bytes.
       dec l               ; es está el límite a horcajadas de 256 bytes aquí.
       inc h               ; siguiente fila de esta celda de carácter.
sprit6 ex de,hl            ; Dirección de pantalla en de.
       ld hl,(tmp0)        ; restaurar la dirección del gráfico.
       ex af,af'           ; restaurar el contador del bucle.
       dec a               ; decrementarlo.
       jp nz,sprit1        ; no alcanzado el borde inferior del sprite, repetir.
       ret                 ; trabajo hecho.
```

```
sprit4 ld de,30            ; el siguiente segmento es de 30 bytes.
       add hl,de           ; añadir a la dirección de la pantalla.
       jp sprit6           ; repetir.
sprit5 ld de,63774         ; menos 1762.
       add hl,de           ; restar 1762 de la dirección física de la pantalla.
       jp sprit6           ; volver al bucle.

scadd  ld a,(xcoord)       ; recuperar la coordenada vertical.
       ld e,a              ; guardarla en e.

; Encuentra línea dentro de la celda.

       and 7               ; línea 0-7 en el cuadrado de caracteres.
       add a,64            ; 64 * 256 = 16384 = inicio de la pantalla.
       ld d,a              ; d = línea * 256.

; Encuentra en que tercio de la pantalla estamos.

       ld a,e              ; restaurar la vertical.
       and 192             ; segmento 0, 1 o 2 multiplicamos por 64.
       rrca                ; dividirlo por 8.
       rrca
       rrca                ; segmento 0-2 multiplicado por 8.
       add a,d             ; sumar a+d obtiene dirección de inicio del segmento.
       ld d,a

; Encuentra la celda de carácter dentro del segmento.

       ld a,e              ; 8 casillas de caracteres por segmento.
       rlca                ; dividir x por 8 y multiplicarlo por 32,
       rlca                ; siguiente cálculo: multiplicar por 4.
       and 224             ; enmascarar los bits que no queremos.
       ld e,a              ; cálculo de la coordenada vertical completo.

; Agregar el elemento horizontal.

       ld a,(ycoord)       ; coordenada y.
       rrca                ; sólo es necesario dividir por 8.
       rrca
       rrca
       and 31              ; cuadrados 0 a 31 a través de la pantalla.
       add a,e             ; añadir al total de la medida.
       ld e,a              ; de = dirección de la pantalla.
       ret

xcoord defb 0              ; coordenada de pantalla.
ycoord defb 0              ; coordenada de pantalla.
tmp0   defw 0              ; espacio de trabajo.
tmp1   defb 0              ; posición vertical temporal.

; tabla de naves enemigas, 8 entradas x 4 bytes cada una.

entab  defb 0,0,0,0
       defb 0,0,0,0
       defb 0,0,0,0
       defb 0,0,0,0
       defb 0,0,0,0
       defb 0,0,0,0
       defb 0,0,0,0
       defb 0,0,0,0
```

descarga el código desde aquí

Enemigos inteligentes

Hasta ahora nos hemos ocupado de drones predecibles, pero ¿queremos dar al jugador la ilusión de que los sprites enemigos piensan por sí mismos? Una vía que podemos utilizar para esto sería para darles una decisión totalmente al azar en el momento de su creación.

Aquí está el código fuente de **Turbomania**, un juego escrito originalmente para la *1K coding competition* (concursos de código en 1K) de 2005. Es muy simple, pero incorpora movimiento puramente aleatorio. Los coches enemigos viajan en una dirección hasta que ya no se pueden mover, a continuación seleccionan otra dirección al azar. Además, un coche puede cambiar de dirección al azar incluso si puede continuar en su dirección actual. Es muy primitivo por supuesto, solo echa un vistazo a la rutina MCAR y verá exactamente lo quiero decir.

```
       org 24576

; Constantes.

YELLOW equ 49               ; atributo de color amarillo normal.
YELLOB equ YELLOW + 64      ; atributo de color amarillo brillante.

; Código principal del juego.

; Borrar la pantalla para dar color verde alrededor de los bordes.

       ld hl,23693          ; variable de sistema para atributos.
       ld (hl),36           ; quiero fondo verde.

waitk  ld a,(23560)         ; leer teclado.
       cp 32                ; ¿pulsado ESPACIO?
       jr nz,waitk          ; no, espera.
       call nexlev          ; jugar,
       jr waitk             ; ESPACIO para reiniciar el juego.


; Borrar los datos por nivel.

nexlev call 3503            ; borrar la pantalla.
       ld hl,rmdat          ; Datos de las salas.
       ld de,rmdat+1
       ld (hl),1            ; configurar un bloque en la sombra.
       ld bc,16             ; longitud de sala menos el primer byte.
       ldir                 ; copiar al resto de la primera fila.
       ld bc,160            ; longitud de la sala menos la primera fila.
       ld (hl),b            ; borrar primer byte.
       ldir                 ; borrar datos de la sala.

; Configurar los bloques predeterminados.

       ld c,15              ; última posición del bloque.
popbl0 ld b,9               ; última fila.
popbl1 call filblk          ; llenar el bloque.
       dec b                ; una columna hacia arriba.
```

```
       jr z,popbl2          ; columna hecha, seguir adelante.
       dec b                ; y de nuevo.
       jr popbl1
popbl2 dec c                ; moverse en la fila.
       jr z,popbl3          ; columna completa, seguir adelante.
       dec c                ; siguiente fila.
       jr popbl0

; Ahora dibujar los bits únicos para este nivel.

popbl3 ld b,7               ; número de bloques para insertar.
popbl5 push bc              ; guardar contador.
       call random          ; obtener un número aleatorio.
       and 6                ; números en el rango 0-6 por favor.
       add a,2              ; cambiar a 2-8.
       ld b,a               ; esa es la columna.
popbl4 call random          ; otro número.
       and 14               ; buscamos números pares 0-12.
       cp 14                ; ¿más alto de lo que queremos?
       jr nc,popbl4         ; si, inténtelo de nuevo.
       inc a                ; colocarlo en el rango 1-13.
       ld c,a               ; esa es la fila.
       call filblk          ; llenar bloque.
popbl6 call random          ; otro número aleatorio.
       and 14               ; Sólo queremos 0-8.
       cp 9                 ; ¿por encima de número que queremos?
       jr nc,popbl6         ; inténtelo de nuevo.
       inc a                ; convertirlo a 1-9.
       ld b,a               ; coordenada vertical.
       call random          ; obtener bloque horizontal.
       and 14               ; par, 0-14.
       ld c,a               ; posición y.
       call filblk          ; llenar en ese cuadrado.
       pop bc               ; restaurar el contador.
       djnz popbl5          ; uno menos que hacer.

       xor a                ; cero.
       ld hl,playi          ; dirección deseada del jugador.
       ld (hl),a            ; dirección por defecto.
       inc hl               ; apuntar a la dirección a presentar.
       ld (hl),a            ; dirección por defecto.
       inc hl               ; siguiente dirección del jugador.
       ld (hl),a            ; dirección por defecto.
       out (254),a          ; establecer el color del borde mientras tanto.
       call atroom          ; mostrar disposición del nivel actual.

       ld hl,168+8*256      ; coordenadas.
       ld (encar2+1),hl     ; establecer la posición del segundo coche.
       ld h,l               ; coordenada y a la derecha.
       ld (encar1+1),hl     ; establecer la posición del primer coche.
       ld l,40              ; x en la parte superior de la pantalla.
       ld (playx),hl        ; iniciar al jugador aquí.
       ld hl,encar1         ; primer coche.
       call scar            ; presentarlo.
       ld hl,encar2         ; segundo coche.
       call scar            ; presentarlo.
       call dplayr          ; mostrar el sprite del jugador.
       call blkcar          ; hacer el coche del jugador de color negro.

; Retardo de dos segundos antes de empezar.
```

```
       ld b,100             ; longitud del retardo.
waitt  halt                 ; esperar una interrupción.
       djnz waitt           ; repetir.

mloop  halt                 ; haz de electrones en la parte superior izquierda.
       call dplayr          ; Eliminar jugador.

; Poner como atributo tinta azul de nuevo.

       call gpatts          ; obtener atributo del jugador.
       defb 17,239,41       ; retirar el fondo verde, añadir fondo y tinta azul.
       call attblk          ; establecer el color de la carretera.

; Mover el coche del jugador.

       ld a,(playd)         ; dirección .
       ld bc,(playx)        ; coordenadas del jugador.
       call movc            ; mover coordenadas.
       ld hl,(dispx)        ; nuevas coordenadas.
       ld (playx),hl        ; establecer una nueva posición del jugador.

; ¿Podemos cambiar de dirección?

       ld a,(playi)         ; dirección deseada del jugador.
       ld bc,(playx)        ; coordenadas del jugador.
       call movc            ; mover coordenadas.
       call z,setpn         ; establecer la nueva dirección del jugador.

; Cambia la dirección.

       ld a,(nplayd)        ; nueva dirección del jugador.
       ld (playd),a         ; fijar la dirección actual.

       call dplayr          ; volver a mostrar en la nueva posición.

; Establecer los atributos de los coches.

       call blkcar          ; hacer el coche del jugador negro.

; Controles.

       ld a,239             ; fila del teclado 6-0 = 61438.
       ld e,1               ; dirección derecha.
       in a,(254)           ; leer teclas.
       rra                  ; ¿jugador mueve a la derecha?
       call nc,setpd        ; sí, establecer la dirección del jugador.
       ld e,3               ; dirección izquierda.
       rra                  ; ¿jugador movió a la izquierda
       call nc,setpd        ; sí, establecer la dirección del jugador.
       ld a,247             ; 63486 es el puerto para la fila 1-5.
       ld e,0               ; dirección hacia arriba.
       in a,(254)           ; leer teclas.
       and 2                ; comprobar segunda tecla en (2).
       call z,setpd         ; establecer la dirección.
       ld a,251             ; 64510 es el puerto para la fila Q-T.
       ld e,2               ; dirección hacia abajo.
       in a,(254)           ; leer teclas.
       and e                ; comprobar segunda tecla del borde (W)..
       call z,setpd         ; establecer la dirección.

; Coches enemigos.
```

```
       ld hl,encar1        ; coche del enemigo 1.
       push hl             ; guardar puntero.
       call procar         ; procesar el coche.
       pop hl              ; recuperar el puntero del coche.
       call coldet         ; comprobar colisiones.
       ld hl,encar2        ; coche del enemigo 2.
       push hl             ; guardar puntero.
       halt                ; sincronizar con pantalla.
       call procar         ; procesar el coche.
       pop hl              ; recuperar el puntero del coche.
       call coldet         ; comprobar colisiones.

; Contar espacio amarillo restante.

       ld hl,22560         ; dirección.
       ld bc,704           ; atributos que contar.
       ld a,YELLOB         ; atributos que estamos buscando.
       cpir                ; contar caracteres.
       ld a,b              ; byte alto del resultado.
       or c                ; combinan con el byte bajo.
       jp z,nexlev         ; nada a la izquierda, ir al siguiente nivel.

; Fin del bucle principal.

       jp mloop

; Coche negro sobre fondo cian.

blkcar call gpatts         ; obtener atributos de los jugadores.
       defb 17,232,40      ; quitar el fondo rojo/tinta azul, añadir fondo azul.

; Establecer los atributos de bloque de 16x16 píxeles.

attblk call attlin         ; pintar línea horizontal.
       call attlin         ; pintar otra línea.
       ld a,c              ; posición vertical.
       and 7               ; ¿está a caballo entre celdas?
       ret z               ; no, entonces no hay tercera línea.

attlin call setatt         ; pintar la carretera.
       call setatt         ; y otra vez.
       ld a,b              ; posición horizontal.
       and 7               ; ¿a caballo entre los bloques?
       jr z,attln0         ; no, dejar tercera celda como está.
       call setatt         ; establecer atributo.
       dec l               ; atrás de nuevo una celda.
attln0 push de             ; conservar los colores.
       ld de,30            ; distancia al siguiente.
       add hl,de           ; puntero a la siguiente fila hacia abajo.
       pop de              ; restaurar máscaras de color.
       ret

; Establecer el atributo de una sola celda.

setatt ld a,(hl)           ; recuperar el atributo de la celda que lo contiene.
       and e               ; eliminar los elementos de color en el registro c.
       or d                ; añadir los de b para formar nuevo color.
       ld (hl),a           ; establecer color.
       inc l               ; siguiente celda.
       ret
```

```
; Detección de colisiones, basada en coordenadas.

coldet call getabc          ; obtener coordenadas.
       ld a,(playx)         ; posición horizontal.
       sub c                ; comparar con el coche x.
       jr nc,coldt0         ; resultado fue positivo.
       neg                  ; era negativo, revertir el signo.
coldt0 cp 16                ; ¿dentro de los 15 píxeles?
       ret nc               ; no hubo colisión.
       ld a,(playy)         ; jugador y.
       sub b                ; comparar con el coche y.
       jr nc,coldt1         ; resultado fue positivo.
       neg                  ; era negativo, revertir el signo.
coldt1 cp 16                ; ¿dentro de los 15 píxeles?
       ret nc               ; no hubo colisión.
       pop de               ; eliminar la dirección de retorno de la pila.
       ret

setpd  ex af,af'
       ld a,e               ; dirección.
       ld (playi),a         ; fijar la dirección prevista.
       ex af,af'
       ret

setpn  ld a,(playi)         ; nueva dirección deseada.
       ld (nplayd),a        ; establecer siguiente dirección.
       ret

; Mover las coordenadas del sprites en la dirección correspondiente.

movc   ld (dispx),bc        ; posición por defecto.
       and a                ; dirección 0.
       jr z,movcu           ; mover hacia arriba.
       dec a                ; dirección 1.
       jr z,movcr           ; mover hacia arriba.
       dec a                ; dirección 2.
       jr z,movcd           ; mover hacia arriba.
movcl  dec b                ; izquierda un píxel.
       dec b                ; izquierda de nuevo.
movc0  call chkpix          ; comprobar los atributos de píxeles.
       ld (dispx),bc        ; nuevas coordenadas.
       ret
movcu  dec c                ; arriba un pixel.
       dec c                ; y de nuevo.
       jr movc0
movcr  inc b                ; derecha un píxel.
       inc b                ; derecha de nuevo.
       jr movc0
movcd  inc c                ; abajo un píxel.
       inc c                ; una vez más.
       jr movc0


; Comprobar los atributos del pixel por colisión.
; Cualquier celda con tinta verde es sólidas.

chkpix call ataddp          ; obtener la dirección del atributo de píxeles.
       and 4                ; comprobar color de la tinta.
       jr nz,chkpx0         ; inválida, bloquear el movimiento.
       inc hl               ; siguiente cuadrado a la derecha.
```

```
       ld a,(hl)            ; obtener atributos.
       and 4                ; comprobar colores de la tinta.
       jr nz,chkpx0         ; inválida, bloquear el movimiento.
       inc hl               ; siguiente cuadrado a la derecha.
       ld a,b               ; posición horizontal.
       and 7                ; ¿a caballo entre las celdas?
       jr z,chkpx1          ; no, mirar abajo entonces.
       ld a,(hl)            ; obtener atributos.
       and 4                ; comprobar color de la tinta.
       jr nz,chkpx0         ; inválida, bloquear el movimiento.
chkpx1 ld de,30             ; distancia hasta la siguiente celda hacia abajo.
       add hl,de            ; apuntar aquí.
       ld a,(hl)            ; obtener atributos.
       and 4                ; comprobar color de la tinta.
       jr nz,chkpx0         ; inválida, bloquear el movimiento.
       inc hl               ; siguiente cuadrado a la derecha.
       ld a,(hl)            ; obtener atributos.
       and 4                ; comprobar color de la tinta.
       jr nz,chkpx0         ; inválida, bloquear el movimiento.
       inc hl               ; siguiente cuadrado a la derecha.
       ld a,b               ; posición horizontal.
       and 7                ; ¿a caballo entre las celdas?
       jr z,chkpx2          ; no, mirar abajo entonces.
       ld a,(hl)            ; obtener atributos.
       and 4                ; comprobar color de la tinta.
       jr nz,chkpx0         ; inválida, bloquear el movimiento.
chkpx2 ld a,c               ; distancia desde la parte superior de la pantalla.
       and 7                ; ¿estamos a caballo entre celdas verticales?
       ret z                ; no, el movimiento es bueno.
       add hl,de            ; apuntar allí.
       ld a,(hl)            ; obtener atributos.
       and 4                ; comprobar color de la tinta.
       jr nz,chkpx0         ; inválida, bloquear el movimiento.
       inc hl               ; siguiente cuadrado a la derecha.
       ld a,(hl)            ; obtener atributos.
       and 4                ; comprobar color de la tinta.
       jr nz,chkpx0         ; inválida, bloquear el movimiento.
       inc hl               ; siguiente cuadrado a la derecha.
       ld a,b               ; posición horizontal.
       and 7                ; ¿a caballo entre las celdas?
       ret z                ; no, el movimiento es correcto.
       ld a,(hl)            ; obtener atributos.
       and 4                ; comprobar color de la tinta.
       jr nz,chkpx0         ; inválida, bloquear el movimiento.
       ret                  ; adelante.

chkpx0 pop de               ; eliminar la dirección de retorno de la pila.
       ret

; Llenar un bloque en el mapa.

; Obtener dirección del bloque.

filblk ld a,b               ; número de fila.
       rlca                 ; multiplicar por 16.
       rlca
       rlca
       rlca
       add a,c              ; añadir al desplazamiento total.
       ld e,a               ; este desplazamiento es pequeño.
       ld d,0               ; no necesita byte alto.
```

```
       ld hl,rmdat          ; dirección de datos de la sala.
       add hl,de            ; añadir al bloque de dirección.

; dirección de bloque está en hl, vamos a llenarlo.

       ld (hl),2            ; poner el bloque en on.
       ld de,16             ; distancia al siguiente bloque de abajo.
       add hl,de            ; apuntar allí.
       ld a,(hl)            ; chequearlo.
       and a                ; ¿está establecido?
       ret nz               ; sí, no sobrescribir.
       ld (hl),1            ; ajustar la sombra.
       ret

; Dibujar una pantalla compuesta enteramente de bloques de atributos.

atroom ld hl,rmdat          ; Datos de la sala.
       ld a,1               ; comenzar en la fila 1.
       ld (dispx),a         ; establecer coordenadas.
       ld b,11              ; contar filas.
atrm0  push bc              ; guardar el contador.
       ld b,15              ; contador de columnas.
       ld a,1               ; número de columna.
       ld (dispy),a         ; ajustado a la izquierda de la pantalla.
atrm1  push bc              ; guardar el contador.
       ld a,(hl)            ; obtener siguiente tipo de bloque.
       push hl              ; guardar dirección de datos.
       rlca                 ; numero de bloque doblado.
       rlca                 ; y de nuevo para múltiplo de 4.
       ld e,a               ; desplazamiento para dirección del bloque.
       ld d,0               ; no necesita byte alto.
       ld hl,blkatt         ; atributos del bloque.
       add hl,de            ; apuntar al bloque que queremos.
       call atadd           ; obtener dirección de la posición de la pantalla.
       ldi                  ; transferir primer bloque.
       ldi                  ; y el segundo.
       ld bc,30             ; distancia hasta la siguiente fila.
       ex de,hl             ; conmutar celda y dirección de pantalla.
       add hl,bc            ; apuntar a la siguiente fila por abajo.
       ex de,hl             ; intercambiarlas de nuevo.
       ldi                  ; hacer la tercera celda.
       ldi                  ; atributo de la cuarta celda.
       ld hl,dispy          ; número de columna.
       inc (hl)             ; moverse a través de una celda.
       inc (hl)             ; y otra.
       pop hl               ; restaurar la dirección de la sala.
       pop bc               ; restablecer contador de columna.
       inc hl               ; apuntar al siguiente bloque.
       djnz atrm1           ; hacer resto de la hilera.
       inc hl               ; saltar un carácter, las líneas son múltiplos 16.
       ld a,(dispx)         ; posición vertical.
       add a,2              ; mirar 2 celdas hacia abajo.
       ld (dispx),a         ; nueva fila.
       pop bc               ; restablecer contador de columna.
       djnz atrm0           ; hacer las filas restantes.
       ret

; Atributos del bloque de fondo.

blkatt defb YELLOB,YELLOB  ; espacio.
       defb YELLOB,YELLOB
```

```
       defb YELLOW,YELLOW  ; espacio sombreado.
       defb YELLOB,YELLOB
       defb 124,68         ; Modelo blanco/negro bandera de cuadros.
       defb 68,124

; Calcular la dirección del atributo de carácter en (dispx, dispy).

atadd  push hl             ; necesitamos preservar el par hl.
       ld hl,(dispx)       ; coordenadas a comprobar, en coordenadas de
carácter.
       add hl,hl           ; multiplicar  x e y por 8.
       add hl,hl
       add hl,hl
       ld b,h              ; copiar coordenada y a b.
       ld c,l              ; poner coordenada x en c.
       call ataddp         ; obtener la dirección del pixel.
       ex de,hl            ; poner la dirección en de.
       pop hl              ; restaurar hl.
       ret


; Obtener atributos del jugador.

gpatts ld bc,(playx)       ; coordenadas del jugador.

; Calcular la dirección del atributo para el píxel en (c, b).

ataddp ld a,c              ; mira primero la vertical.
       rlca                ; divida por 64.
       rlca                ; mas rápido que 6 operaciones rrca.
       ld l,a              ; almacenar en el registro l por ahora.
       and 3               ; enmascarar para encontrar el segmento.
       add a,88            ; los atributos comienzan a partir de 88*256=22528.
       ld h,a              ; así está ordenado el byte alto.
       ld a,l              ; vertical/64 es igual que vertical*4.
       and 224             ; buscamos un múltiplo de 32.
       ld l,a              ; elemento vertical calculado.
       ld a,b              ; obtener la posición horizontal.
       rra                 ; dividir por 8.
       rra
       rra
       and 31              ; queremos el resultado en el rango 0-31.
       add a,l             ; añadir a byte bajo existente.
       ld l,a              ; completado el byte bajo.
       ld a,(hl)           ; obtener contenido de la celda.
       ret                 ; dirección del atributo ahora en hl.

; Mover el coche - cambio de dirección requerida.

mcarcd ld a,(hl)           ; dirección actual.
       inc a               ; gira en el sentido de las manecillas del reloj.
       and 3               ; sólo 4 direcciones.
       ld (hl),a           ; nueva dirección.

; Mover un coche enemigo.

mcar   push hl             ; preservar puntero al coche.
       call getabc         ; recuperar coordenadas y dirección.
       call movc           ; mover el coche.
       pop hl              ; refrescar puntero del coche
       jr nz,mcarcd        ; no se puede moverse mas, dar la vuelta.
```

```
       inc hl               ; apuntar a x.
       ld a,c               ; guardar posición x en c.
       ld (hl),a            ; posición x.
       inc hl               ; apuntar a y.
       ld (hl),b            ; nueva colocación.
       or b                 ; combinar ambos.
       and 31               ; encontrar celdas de posición a caballo.
       cp 8                 ; ¿estamos en un punto de giro válido?
       ret nz               ; no, no se puede cambiar de dirección.
       ld a,r               ; falso número aleatorio.
       cp 23                ; compruebe que está por debajo de este valor.
       ret nc               ; no lo está, no cambia.
       push hl              ; guardar puntero del coche.
       call random          ; obtener número aleatorio.
       pop hl               ; restaurar coche.
       dec hl               ; volver a la coordenada x.
       dec hl               ; volver de nuevo a la dirección.
       and 3                ; dirección en el rango de  0-3.
       ld (hl),a            ; nueva dirección.
       ret

; Obtener coordenadas de automóviles y su dirección.

getabc ld a,(hl)            ; obtener dirección.
       inc hl               ; apuntar a la posición x.
       ld c,(hl)            ; coordenada x.
       inc hl               ; apuntar a y.
       ld b,(hl)            ; posición y.
       ret


; Procesar coche en el punto hl.

procar push hl              ; guardar puntero.
       push hl              ; guardar puntero.
       call scar            ; Eliminar coche.
       pop hl               ; restablecer puntero a coche.
       call mcar            ; mover coche.
       pop hl               ; restablecer puntero a coche.


; Mostrar coche enemigo.

scar   call getabc          ; obtener coordenadas y dirección.
       jr dplay0

; Presentar sprite del jugador.

dplayr ld bc,(playx)        ; coordenadas del jugador.
       ld a,(playd)         ; dirección del jugador.
dplay0 rrca                 ; multiplicar por 32.
       rrca
       rrca
       ld e,a               ; sprite * 32 en el byte bajo.
       ld d,0               ; sin byte alto.
       ld hl,cargfx         ; gráfico del coche.
       add hl,de            ; añadir desplazamiento al sprite.
       jr sprite            ; mostrar el sprite.


; Esta es la rutina de sprites y espera las coordenadas en la forma (c, b),
```

```
; donde c es la coordenada vertical desde la parte superior de la pantalla
; (0-176), y b es la coordenada horizontal desde la izquierda de la pantalla
; (0 a 240). Los datos del sprite se almacenan como se espera en su forma no
; desplazada ya que esta rutina se encarga de todo el desplazamiento por sí
; misma. Esto significa que direccionar el sprite no es especialmente rápido,
; pero los gráficos sólo ocupan 1/8 del espacio que requerirían en forma
; pre-desplazada.

; La entrada HL debe apuntar a los datos del sprite sin desplazamiento.

sprit7 xor 7               ; complementa los últimos 3 bits.
       inc a               ; ¡agrega uno por suerte!
sprit3 rl d                ; rotar izquierda...
       rl c                ; ...en el centro del byte...
       rl e                ; ...y a la izquierda de la celda de carácter.
       dec a               ; contar los cambios que hemos hecho.
       jr nz,sprit3        ; regresar hasta que los movimientos estén completos.

; Línea de la imagen de sprite ahora en e+c+d, lo necesitamos en forma c+d+e

       ld a,e              ; borde izquierdo de la imagen está en e.
       ld e,d              ; poner borde derecho en su lugar.
       ld d,c              ; bit central va en d
       ld c,a              ; y el borde izquierdo de nuevo en c.
       jr sprit0           ; hemos hecho el cambio para transferir a la
pantalla.

sprite ld (dispx),bc       ; guardar ahora coordenadas en dispx.
       call scadd          ; calcular dirección de la pantalla.
       ld a,16             ; altura del sprite en pixeles.
sprit1 ex af,af'           ; guardar el contador de bucles.
       push de             ; guardar dirección de pantalla.
       ld c,(hl)           ; primer gráfico del sprite.
       inc hl              ; incrementar el puntero de datos del sprite.
       ld d,(hl)           ; siguiente bit de la imagen del sprite.
       inc hl              ; apuntar a la siguiente fila de datos del sprite.
       ld (sprtmp),hl      ; guardar para más adelante.
       ld e,0              ; byte derecho en blanco por ahora.
       ld a,b              ; b guarda la posición y.
       and 7               ; ¿estamos a caballo entre celdas de caracteres?
       jr z,sprit0         ; no estamos a caballo, no molesta al desplazamiento.
       cp 5                ; ¿necesitamos 5 o más desplazamientos a la derecha?
       jr nc,sprit7        ; sí, desplazar a la izquierda que es más rápido.
       and a               ; hay, la bandera de acarreo se establece.
sprit2 rr c                ; rotar a la izquierda el byte derecho...
       rr d                ; ...Hasta el byte medio...
       rr e                ; ...la byte derecho.
       dec a               ; un turno menos que hacer.
       jr nz,sprit2        ; repetir hasta que todos los turnos estén completos.
sprit0 pop hl              ; sacar la dirección de la pantalla en la pila.
       ld a,(hl)           ; ya está lista.
       xor c               ; fusionar con los datos de la imagen.
       ld (hl),a           ; colocar en la pantalla.
       inc l               ; siguiente celda de carácter por la derecha.
       ld a,(hl)           ; ya estaba antes.
       xor d               ; fusionarse con el centro de la imagen.
       ld (hl),a           ; poner de nuevo en la pantalla.
       inc l               ; siguiente bit del área de la pantalla.
       ld a,(hl)           ; lo que ya está allí.
       xor e               ; borde derecho de los datos de imagen del sprite.
       ld (hl),a           ; poner en pantalla.
```

```
       ld a,(dispx)         ; coordenada vertical temporal.
       inc a                ; siguiente línea de abajo.
       ld (dispx),a         ; almacenar nueva posición.
       and 63               ; ¿nos movemos al siguiente tercio de pantalla?
       jr z,sprit4          ; sí, encontrar el próximo segmento.
       and 7                ; ¿entrando en celda de carácter siguiente?
       jr z,sprit5          ; Sí, encuentre siguiente fila.
       dec l                ; izquierda 2 bytes.
       dec l                ; es está el límite a horcajadas de 256 bytes aquí.
       inc h                ; siguiente fila de esta celda de carácter.
sprit6 ex de,hl             ; Dirección de pantalla en de.
       ld hl,(sprtmp)       ; restaurar la dirección del gráfico.
       ex af,af'            ; restaurar el contador del bucle.
       dec a                ; decrementarlo.
       jp nz,sprit1         ; no alcanzado el borde inferior del sprite, repetir.
       ret                  ; trabajo hecho.
sprit4 ld de,30             ; el siguiente segmento es de 30 bytes.
       add hl,de            ; añadir a la dirección de la pantalla.
       jp sprit6            ; repetir.
sprit5 ld de,63774          ; menos 1762.
       add hl,de            ; restar 1762 de la dirección física de la pantalla.
       jp sprit6            ; volver al bucle.

; Esta rutina devuelve la dirección de pantalla de (c, b) en de.

scadd  ld a,c               ; obtener la posición vertical.
       and 7                ; línea 0-7 en el cuadro del carácter.
       add a,64             ; 64 * 256 = 16384 (inicio de la pantalla)
       ld d,a               ; línea * 256.
       ld a,c               ; obtener verticales de nuevo.
       rrca                 ; multiplicar por 32.
       rrca
       rrca
       and 24               ; byte alto del desplazamiento de segmento.
       add a,d              ; añadir a byte alto de pantalla existente.
       ld d,a               ; ese es el byte alto ordenado.
       ld a,c               ; 8 casillas de carácter por segmento
       rlca                 ; 8 píxeles por celda, multiplicado por 4 = 32.
       rlca                 ; celda x 32 da la posición dentro del segmento.
       and 224              ; asegurarse de que es un múltiplo de 32.
       ld e,a               ; cálculo de coordenada vertical hecho.
       ld a,b               ; coordenada y.
       rrca                 ; Sólo es necesario dividir por 8.
       rrca
       rrca
       and 31               ; cuadrados 0 - 31 a través de la pantalla.
       add a,e              ; añadirse al total de la medida.
       ld e,a               ; hl = dirección de la pantalla.
       ret

; Generador de números seudo-aleatorio.
; Llevando un puntero a través de la ROM (a partir de una semillas),
; devuelve el contenido del byte en esa ubicación.

random ld hl,(seed)         ; puntero a la ROM.
       res 5,h              ; mantenerse dentro de los primeros 8K de ROM.
       ld a,(hl)            ; obtener el número "aleatorio" de esa ubicación.
       xor l                ; más aleatoriedad.
       inc hl               ; incrementar el puntero.
       ld (seed),hl         ; nueva posición.
       ret
```

```
; Datos gráficos del Sprite.
; Primero apunta subiendo.

cargfx defb 49,140,123,222,123,222,127,254,55,236,15,240,31,248,30,120
       defb 29,184,108,54,246,111,255,255,247,239,246,111,103,230,3,192

; Segunda imagen apunta a la derecha.

       defb 60,0,126,14,126,31,61,223,11,238,127,248,252,254,217,127
       defb 217,127,252,254,127,248,11,238,61,223,126,31,126,14,60,0

; La tercera apunta hacia abajo.

       defb 3,192,103,230,246,111,247,239,255,255,246,111,108,54,29,184
       defb 30,120,31,248,15,240,55,236,127,254,123,222,123,222,49,140

; La última el coche apunta a la izquierda.

       defb 0,60,112,126,248,126,251,188,119,208,31,254,127,63,254,155
       defb 254,155,127,63,31,254,119,208,251,188,248,126,112,126,0,60

; Variables usadas por el juego.

       org 32768

playi  equ $                ; dirección deseada cuando el giro es posible.
playd  equ playi+1          ; dirección actual del jugador.
nplayd equ playd+1          ; siguiente dirección jugador.
playx  equ nplayd+1         ; coordenada x del jugador
playy  equ playx+1          ; coordenada y del jugador.

encar1 equ playy+1          ; coche enemigo 1.
encar2 equ encar1+3         ; coche enemigo 2.

dispx  equ encar2+3         ; coordenadas de uso general.
dispy  equ dispx+1
seed   equ dispy+1          ; semilla de números aleatorios.
sprtmp equ seed+2           ; dirección temporal del sprite.
termin equ sprtmp+2         ; fin de variables.

rmdat  equ 49152
```

descarga el código desde aquí

Si has ensamblado este juego y lo has probado, te darás cuenta de que rápidamente se vuelve aburrido. Es muy fácil mantenerse fuera del alcance de los enemigos cubriendo un lado de la pista, y luego esperar hasta que se muevan y cubrir el otro lado. Este algoritmo no tiene aspecto de cazador-asesino por lo que el jugador nunca es perseguido. Es más, esta rutina es de simples vehículos que no cambian de dirección sin advertencia. En la mayoría de juegos esto sólo es aceptable si un sprite llega a un callejón sin salida y no se puede mover en cualquier otra dirección.

Tal vez debemos en su lugar escribir rutinas en que los aliens interactúan con el jugador, y lo persiguen. Así, el algoritmo más básico sería seguir una línea comprobando las coordenadas base x/y, y mover el sprite del alien hacia el jugador. La rutina siguiente muestra cómo podría lograrse este objetivo, la rutina buscadora de blancos **almov** es la que mueve el sprite alrededor del jugador. Intente guiar al bloque número 1 por la pantalla con las teclas A, S, D y F, y el bloque número 2 te seguirá alrededor de la pantalla. Sin embargo, al hacer esto pronto descubrimos el defecto básico con este tipo de persecución, es muy fácil atrapar el sprite enemigo en una esquina porque la rutina no es lo suficientemente inteligente para moverse hacia atrás con el fin de conseguir rodear los obstáculos.

```
; Aleatoriamente cubrir la pantalla con bloques amarillos.

       ld de,1000          ; dirección en la ROM.
       ld b,64             ; número de celdas de color.
yello0 push bc             ; almacenas el registro.
       ld a,(de)           ; obtener la primera coordenada al azar.
       and 127             ; la mitad de la altura de la pantalla.
       add a,32            ; al menos 32 píxeles hacia abajo.
       ld c,a              ; coordenada x.
       inc de              ; siguiente byte de ROM.
       ld a,(de)           ; recuperar valor.
       inc de              ; siguiente byte de ROM.
       ld b,a              ; coordenada y.
       call ataddp         ; encuentra la dirección del atributo.
       ld (hl),48          ; establecer atributos.
       pop bc              ; restaurar el contador del bucle.
       djnz yello0         ; repetir varias veces.

       ld ix,aldat         ; datos del alien.
       call dal            ; mostrar alien.
       call dpl            ; presentar jugador.

mloop  halt                ; esperar al haz de electrones.
       call dal            ; eliminar alien.
       call almov          ; movimiento del alien.
       call dal            ; mostrar alien.
       halt                ; esperar al haz de electrones.
       call dpl            ; eliminar jugador.
       call plcon          ; controles del jugador.
       call dpl            ; mostrar al jugador.
       jp mloop            ; volver al inicio del bucle principal.

aldat  defb 0,0,0          ; datos del alien.

; Visualizar/eliminar alien.

dal    ld c,(ix)           ; posición vertical.
       ld b,(ix+1)         ; posición horizontal.
       ld (xcoord),bc      ; establecer las coordenadas del sprite.
       ld hl,algfx         ; gráfico del alien.
       jp sprite           ; XOR del sprite con la pantalla.

; Visualizar/eliminar sprite del jugador.

dpl    ld bc,(playx)       ; coordenadas.
       ld (xcoord),bc      ; establecer las coordenadas de pantalla.
       ld hl,plgfx         ; gráfico del jugador.
```

```
       jp sprite           ; xor del sprite dentro o fuera de la pantalla.

; control del jugador.

plcon  ld bc,65022         ; puerto para la fila del teclado.
       in a,(c)            ; leer teclado.
       ld b,a              ; almacenar el resultado en el registro b.
       rr b                ; verificación la tecla más externa.
       call nc,mpl         ; jugador a la izquierda.
       rr b                ; comprobar siguiente tecla.
       call nc,mpr         ; jugador de la derecha.
       rr b                ; comprobar siguiente tecla.
       call nc,mpd         ; jugador abajo.
       rr b                ; comprobar siguiente tecla.
       call nc,mpu         ; jugador arriba.
       ret
mpl    ld hl,playy         ; coordenada.
       ld a,(hl)           ; comprobar el valor.
       and a               ; ¿en el borde de la pantalla?
       ret z               ; sí, no se puede mover en esa dirección.
       sub 2               ; moverse 2 pixeles.
       ld (hl),a           ; nuevo ajuste.
       ret
mpr    ld hl,playy         ; coordenada.
       ld a,(hl)           ; comprobar el valor.
       cp 240              ; ¿en el borde de la pantalla?
       ret z               ; sí, no se puede mover en esa dirección.
       add a,2             ; moverse 2 pixeles.
       ld (hl),a           ; nuevo ajuste.
       ret
mpu    ld hl,playx         ; coordenada.
       ld a,(hl)           ; comprobar el valor.
       and a               ; ¿en el borde de la pantalla?
       ret z               ; sí, no se puede mover en esa dirección.
       sub 2               ; moverse 2 pixeles.
       ld (hl),a           ; nuevo ajuste.
       ret
mpd    ld hl,playx         ; coordenada.
       ld a,(hl)           ; comprobar el valor.
       cp 176              ; ¿en el borde de la pantalla?
       ret z               ; sí, no se puede mover en esa dirección.
       add a,2             ; moverse 2 pixeles.
       ld (hl),a           ; nuevo ajuste.
       ret


; Rutina de movimiento del alien.

almov  ld a,(playx)        ; coordenada x del jugador.
       ld c,(ix)           ; alien x.
       ld b,(ix+1)         ; alien y.
       cp c                ; revisar x del alien
       jr z,alv0           ; son iguales, seguir en horizontal.
       jr c,alu            ; el alien está más abajo, moverlo hacia arriba.
ald    inc c               ; el alien está más arriba, moverlo hacia abajo.
       jr alv0             ; Ahora comprobar la posición de las paredes.
alu    dec c               ; mover hacia abajo.
alv0   call alchk          ; comprobar atributos.
       cp 56               ; ¿están bien?
       jr z,alv1           ; Sí, establecer la coordenada x.
       ld c,(ix)           ; restaurar la anterior coordenada x.
```

```
       jr alh              ; ahora ir en horizontal.
alv1   ld (ix),c           ; nueva coordenada x.
alh    ld a,(playy)        ; horizontal del jugador.
       cp b                ; comprobar horizontal del alien.
       jr z,alok           ; son iguales, verificar la colisión.
       jr c,all            ; alien a la derecha, mover a la izquierda.
alr    inc b               ; alien a la izquierda, mover a la derecha.
       jr alok             ; comprobar las paredes.
all    dec b               ; mover a la derecha.
alok   call alchk          ; comprobar atributos.
       cp 56               ; ¿están bien?
       ret nz              ; no, establecer la nueva coordenada y.
       ld (ix+1),b         ; establecer nueva y.
       ret


; Comprueba atributos en la posición de alien (c,b).

alchk  call ataddp         ; conseguir dirección del atributo.
       ld a,3              ; celda de arriba.
alchk0 ex af,af'           ; guardar el contador de bucles.
       ld a,(hl)           ; verificar color de la celda.
       cp 56               ; ¿es negro sobre blanco?
       ret nz              ; no, no se puede mover aquí.
       inc hl              ; celda de la derecha.
       ld a,(hl)           ; verificar color de la celda.
       cp 56               ; ¿es negro sobre blanco?
       ret nz              ; no, no se puede mover aquí.
       inc hl              ; celda de la izquierda.
       ld a,(hl)           ; verificar color de la celda.
       cp 56               ; ¿es negro sobre blanco?
       ret nz              ; no, no se puede mover aquí.
       ld de,30            ; distancia hasta la siguiente celda hacia abajo.
       add hl,de           ; mira aquí.
       ex af,af'           ; contador de alturas.
       dec a               ; una menos a la que ir.
       jr nz,alchk0        ; repetir para todas las filas.
       ld (ix),c           ; establecer nueva x.
       ld (ix+1),b         ; establecer nueva y.
       ret

; Calcular la dirección del atributo para el píxel en (c,b).

ataddp ld a,c              ; Mira primero en vertical.
       rlca                ; dividir por 64.
       rlca                ; más rápido que 6 instrucciones rrca.
       ld l,a              ; almacenar en el registro l por ahora.
       and 3               ; enmascarar para encontrar segmento.
       add a,88            ; atributos comienzan a partir de 88*256=22528.
       ld h,a              ; ese es nuestro byte alto ordenado.
       ld a,l              ; vertical/64, es lo mismo que vertical*4.
       and 224             ; deseamos un múltiplo de  32.
       ld l,a              ; elemento vertical calculado.
       ld a,b              ; obtener la posición horizontal.
       rra                 ; dividir por 8.
       rra
       rra
       and 31              ; queremos dar lugar a rango 0-31.
       add a,l             ; añadir al byte bajo existente.
       ld l,a              ; tenemos el byte bajo hecho.
       ret                 ; dirección de atributo ahora en hl.
```

```
playx  defb 80             ; coordenadas del jugador.
playy  defb 120
xcoord defb 0              ; coordenadas generales multi propósito.
ycoord defb 0

; Rutina de sprites desplazados.

sprit7 xor 7               ; complementa los últimos 3 bits.
       inc a               ; ¡agrega uno por suerte!
sprit3 rl d                ; rotar izquierda...
       rl c                ; ...en el centro del byte...
       rl e                ; ...y a la izquierda de la celda de carácter.
       dec a               ; contar los cambios que hemos hecho.
       jr nz,sprit3        ; regresar hasta que los movimientos estén completos.

; Línea de la imagen de sprite ahora en e+c+d, lo necesitamos en forma c+d+e

       ld a,e              ; borde izquierdo de la imagen está en e.
       ld e,d              ; poner borde derecho en su lugar.
       ld d,c              ; bit central va en d
       ld c,a              ; y el borde izquierdo de nuevo en c.
       jr sprit0           ; hemos hecho el cambio para transferir a la
pantalla.

sprite ld a,(xcoord)       ; dibuja el sprite (hl).
       ld (tmp1),a         ; guardar vertical.
       call scadd          ; calcular dirección de la pantalla.
       ld a,16             ; altura del sprite en pixeles.
sprit1 ex af,af'           ; guardar el contador de bucles.
       push de             ; guardar dirección de pantalla.
       ld c,(hl)           ; primer gráfico del sprite.
       inc hl              ; incrementar el puntero de datos del sprite.
       ld d,(hl)           ; siguiente bit de la imagen del sprite.
       inc hl              ; apuntar a la siguiente fila de datos del sprite.
       ld (tmp0),hl        ; guardar en tmp0 para más adelante.
       ld e,0              ; byte derecho en blanco por ahora.
       ld a,b              ; b guarda la posición y.
       and 7               ; ¿estamos a caballo entre celdas de caracteres?
       jr z,sprit0         ; no estamos a caballo, no molesta al desplazamiento.
       cp 5                ; ¿necesitamos 5 o más desplazamientos a la derecha?
       jr nc,sprit7        ; sí, desplazar a la izquierda que es más rápido.
       and a               ; hay, la bandera de acarreo se establece.
sprit2 rr c                ; rotar a la izquierda el byte derecho...
       rr d                ; ...Hasta el byte medio...
       rr e                ; ...la byte derecho.
       dec a               ; un turno menos que hacer.
       jr nz,sprit2        ; repetir hasta que todos los turnos estén completos.
sprit0 pop hl              ; sacar la dirección de la pantalla en la pila.
       ld a,(hl)           ; ya está lista.
       xor c               ; fusionar con los datos de la imagen.
       ld (hl),a           ; colocar en la pantalla.
       inc l               ; siguiente celda de carácter por la derecha.
       ld a,(hl)           ; ya estaba antes.
       xor d               ; fusionarse con el centro de la imagen.
       ld (hl),a           ; poner de nuevo en la pantalla.
       inc hl              ; siguiente bit del área de la pantalla.
       ld a,(hl)           ; lo que ya está allí.
       xor e               ; borde derecho de los datos de imagen del sprite.
       ld (hl),a           ; poner en pantalla.
```

```
       ld a,(tmp1)          ; coordenada vertical temporal.
       inc a                ; siguiente línea de abajo.
       ld (tmp1),a          ; almacenar nueva posición.
       and 63               ; ¿nos movemos al siguiente tercio de pantalla?
       jr z,sprit4          ; sí, encontrar el próximo segmento.
       and 7                ; ¿entrando en celda de carácter siguiente?
       jr z,sprit5          ; Sí, encuentre siguiente fila.
       dec hl               ; izquierda 2 bytes.
       dec l                ; es está el límite a horcajadas de 256 bytes aquí.
       inc h                ; siguiente fila de esta celda de carácter.
sprit6 ex de,hl             ; Dirección de pantalla en de.
       ld hl,(tmp0)         ; restaurar la dirección del gráfico.
       ex af,af'            ; restaurar el contador del bucle.
       dec a                ; decrementarlo.
       jp nz,sprit1         ; no alcanzado el borde inferior del sprite, repetir.
       ret                  ; trabajo hecho.
sprit4 ld de,30             ; el siguiente segmento es de 30 bytes.
       add hl,de            ; añadir a la dirección de la pantalla.
       jp sprit6            ; repetir.
sprit5 ld de,63774          ; menos 1762.
       add hl,de            ; restar 1762 de la dirección física de la pantalla.
       jp sprit6            ; volver al bucle.

scadd  ld a,(xcoord)        ; recuperar la coordenada vertical.
       ld e,a               ; guardarla en e.

; Encuentra la línea dentro de la celda.

       and 7                ; línea 0-7 en el cuadro del carácter.
       add a,64             ; 64 * 256 = 16384 = inicio de la pantalla.
       ld d,a               ; línea * 256.

; Encuentra en que tercio de pantalla estamos.

       ld a,e               ; restaurar la vertical.
       and 192              ; segmento 0, 1 o 2 multiplicado por 64.
       rrca                 ; dividirlo por 8.
       rrca
       rrca                 ; segmento 0-2 multiplicado por 8.
       add a,d              ; añadir a d da la dirección inicial del segmento.
       ld d,a

; Encuentra celda de carácter dentro del segmento.

       ld a,e               ; 8 casillas de caracteres por segmento.
       rlca                 ; dividir por 8 y multiplicar por 32,
       rlca                 ; cálculo neto: multiplicar por 4.
       and 224              ; enmascarar los bits que no queremos.
       ld e,a               ; cálculo de coordenada vertical hecho.

; Añadir el elemento horizontal.

       ld a,(ycoord)        ; coordenada y.
       rrca                 ; sólo es necesario dividir por 8.
       rrca
       rrca
       and 31               ; cuadrados 0 - 31 por la pantalla.
       add a,e              ; añadirse al total de la medida.
       ld e,a               ; de = dirección de la pantalla.
       ret
```

```
tmp0   defw 0
tmp1   defb 0


plgfx  defb 127,254,255,255,254,127,252,127,248,127,248,127,254,127,254,127
       defb 254,127,254,127,254,127,254,127,248,31,248,31,255,255,127,254
algfx  defb 127,254,254,63,248,15,240,135,227,231,231,231,255,199,255,15
       defb 252,31,248,127,241,255,227,255,224,7,224,7,255,255,127,254
```

descarga el código desde aquí

Las mejores rutinas de movimiento de aliens usan una combinación de elementos aleatorios y algoritmos de cazador-asesino. Para superar el problema en el listado superior necesitamos un nuevo indicador adicional para indicar el estado actual del enemigo o en su caso su dirección. Podemos mover el sprite a lo largo de una dirección determinada hasta que sea posible y cambiar el curso vertical u horizontal, con lo cual se selecciona una nueva dirección dependiendo de la posición del jugador. Sin embargo, en caso de que no sea posible mover en la dirección deseada iremos en la dirección opuesta en su lugar. Utilizando este método un sprite puede encontrar su propio camino en la mayoría de los laberintos sin colgarse con demasiada frecuencia. De hecho, para garantizar absolutamente que el sprite no permanecerá atrapados podemos añadir un elemento al azar de modo que cada cierto tiempo la nueva dirección es elegido de forma aleatoria en lugar de la diferencia en las coordenadas x e y

Subiendo la dificultad por niveles

La ponderación aplicada a la decisión sobre los cambios de dirección determinarán los niveles de inteligencia de los sprites. Si la nueva dirección tiene una probabilidad del 90% de ser elegido de manera aleatoria y una probabilidad del 10% sobre la base de coordenadas, el alien paseará sin rumbo por un tiempo y sólo se acerca cuando el jugador se mueve lento. Así, una decisión aleatoria a veces puede ser la más adecuada cuando persigue al jugador. Un alien en una pantalla más difícil podría tener una probabilidad del 60% de elegir una nueva dirección al azar, y un 40% de posibilidades de elegir la dirección en base a la posición relativa de jugador. Este alien hará un seguimiento un poco más de cercano al jugador. Ajustando estos niveles porcentuales, es posible determinar los niveles de dificultad a lo largo de un partido y asegurar una transición suave desde el las más simples pantallas del inicio de la partida a niveles finales diabólicamente difíciles.

Capítulo 12: Temporización

El tiempo lo es todo. Un juego se arruina fácilmente si se ejecuta demasiado rápido o demasiado lento. La mayoría de los juegos de Spectrum se ejecutarán con demasiada rapidez si lo único que hacen es manipular unos sprites, y necesitan ser ralentizados.

La instrucción Halt

Medimos la velocidad de un juego de Spectrum por la cantidad de tiempo que tarda un recorrido completo del bucle principal, incluyendo todos los trabajos realizados por las rutinas llamadas dentro de este bucle. La forma más sencilla de introducir un retardo es insertar instrucciones **halt**, que espera por una interrupción, en ciertos puntos del bucle principal para que espere hasta una interrupción. Como el Spectrum genera 50 interrupciones por segundo, esto significa que el bucle principal que tenga 1, 2 ó 3 de tales pausas se ejecutará a 50, 25 o 17 cuadros por segundo respectivamente, en tanto que el resto de procesamiento no ocupa más de un cuadro para completarse. En términos generales, no es buena idea tener el sprite del jugador moviendo más lentamente de 17 cuadros por segundo.

En realidad, la instrucción **halt** puede ser muy útil. En efecto, espera a que la línea de exploración del televisión llegue al final de la pantalla. Esto significa que un buen momento para borrar, mover y volver a mostrar un sprite es inmediatamente después de un **halt**, porque la línea de exploración no va a encontrarse con la imagen y no hay posibilidad de parpadeo. Si tienes un panel de estado de tu juego en la parte superior de la pantalla, esto significa que hay aún más tiempo para que la línea de exploración viaje antes de que alcance la zona de sprites, y a menudo puedes manipular un par de sprites después de una interrupción sin mucho peligro de parpadeo.

La instrucción **halt** también se puede utilizar en un bucle para hacer una pausa durante períodos más largos. El siguiente código hará una pausa de 100 cincuentavos de segundo (lo que son dos segundos).

```
       ld b,100            ; duración de la pausa.
delay  halt                ; esperar una interrupción.
       djnz delay          ; repetir.
```

El reloj del Spectrum y la rutina Vsync

Por desgracia, **halt** es un instrumento sin punta. Siempre espera a la siguiente interrupción independientemente del tiempo que quede para la siguiente. Imagina una situación en la que el bucle principal tarda 3/4 del tiempo del cuadro para hacer su procesamiento la mayor parte del tiempo, pero de vez en cuando tiene períodos en los que hay un procesamiento adicional que tarda medio tiempo de cuadro adicional. En estas circunstancias, un **halt** mantendrá el juego a la constantes de 50 cuadrod por segundo la mayoría de las veces, pero cuando el procesamiento

adicional entra en acción, la primera interrupción ha pasado y **halt** esperará hasta la siguiente, lo que significa que el juego se ralentiza a 25 cuadros por segundo periódicamente.

Hay una manera de evitar este problema, consiste en contar el número de cuadros que han transcurrido desde la última iteración del bucle principal. En el Spectrum, la rutina de servicio de interrupción de la ROM actualiza el contador de cuadros de 24 bits del Spectrum 50 veces por segundo, y tambien hace otras cosas. Este contador es almacenado en las variables del sistema en la dirección 23672, por lo que mediante la comprobación de esta ubicación una vez en cada iteración del bucle, podemos saber cuántas interrupciones se han producido desde la última vez que estuvimos en ese mismo punto. Naturalmente, si quieres escribir tu propia rutina de manejo de interrupción, puedes utilizar o bien en primer lugar **rst 56** para actualizar el reloj, o bien incrementar un contador de cuadros por ti mismo si deseas utilizar este método.

La siguiente rutina **vsync** está diseñada para estabilizar un juego y hacerlo funcionar a un nivel más o menos constantes de 25 cuadros por segundo:

```
wait   ld hl,pretim        ; carga el temporizador anterior.
       ld a,(23672)        ; carga el temporizador actual.
       sub (hl)            ; diferencia entre los dos.
       cp 2                ; ¿ya han transcurrido dos cuadros?
       jr nc,wait0         ; sí, no más demora.
       jp wait
wait0  ld a,(23672)        ; carga el temporizador actual.
       ld (hl),a           ; guardar su valor como anterior.
       ret
pretim defb 0
```

En lugar de simplemente esperar en un bucle, se podría realizar alguna procesamiento adicional no esencial. Este es un buen punto en el que poner cualquier efecto de sonido del altavoz. Una buena idea es cuando lo necesites establecer un byte para indicar el tipo de efecto de sonido a reproducir, y a continuación comprobar este octeto en tu rutina **vsync** y llamar a la rutina correspondiente al efecto de sonido. Tus rutinas de efectos de sonido también necesitarán controles periódicos para ver si el contador de cuadros ha llegado a su fin, y salir cuando lo ha hecho.

Hay otras cosas que tal vez quiera hacer con este tiempo de CPU. Yo a veces renuevo mis sprites en la tabla que los mantienen, cambiando el orden en que se muestran en cada bucle para ayudar a prevenir el parpadeo.

Semilla de Números Aleatorios

El contador de tramas del Spectrum es útil para otra cosa: se puede utilizar para inicializar la semilla de los números aleatorios. Utilizando el generador de números aleatorios propuesto en el capítulo de números aleatorios, podemos hacer esto:

```
       ld a,(23672)        ; temporizador actual.
       ld (seed),a         ; establece primer byte de la semilla de aleatorios.
```

Esto va bien si estamos trabajando con el hardware real y nos asegurará que un juego no comience con la misma secuencia de números aleatorios cada vez que se juega. Por desgracia, los autores de los emuladores tienen la mala costumbre de cargar automáticamente los archivos de cinta una vez abiertos, una práctica que no sólo hace difícil el desarrollo, da lugar a que la máquina esté siempre en el mismo estado cada vez que el juego se cargue, es decir, los números aleatorios pueden seguir la misma secuencia cada vez que se juega a ese juego. La solución para el programador de juegos es esperar a que se pulse una tecla tan pronto como nuestro juego ha cargado, después de lo cual podemos configurar nuestra semilla. Esto introduce un elemento humano y asegura que el generador de números aleatorios es diferente cada vez.

Capítulo 13: Doble Buffer

Hasta ahora hemos dibujado todos nuestros gráficos directamente en la pantalla, por razones de velocidad y simplicidad. Sin embargo hay una desventaja importante de este método: si la línea de exploración del televisión está cubriendo el área de pantalla en particular en la que se va a suprimir o volver a dibujar nuestra imagen, aparecerán nuestros gráficos con parpadeo. Por desgracia, en el Spectrum no hay una manera fácil de saber donde está la línea de exploración en un momento dado, así que tenemos que encontrar una manera de evitar esto. Un método que funciona bien es borrar y volver a dibujar todos los sprites inmediatamente después de una instrucción **halt**, antes de que el haz de exploración tenga oportunidad de ponerse al día para seguir dibujando la imagen. La desventaja de este método es que nuestro código para los sprites tiene que ser bastante rápido, e incluso en ese caso no es recomendable eliminar y volver a dibujar más de dos sprites por cuadro porque para entonces el haz de exploración estarán ya bajo el borde superior y en el área de la pantalla. Por supuesto, colocar el panel de estado en la parte superior de la pantalla podría dar un poco más de tiempo para dibujar nuestros gráficos, y si el juego corre a 25 fotogramas por segundo podríamos emplear una segunda instrucción **halt** y maniobrar otro par de sprites inmediatamente después. En última instancia, llega un punto en el que esto se rompe. Si nuestros gráficos van a tomar un poco más de tiempo para dibujarse, necesitamos otra manera de ocultar el proceso al jugador, necesitamos usar una segunda pantalla de búfer. Esto significa que todo el trabajo involucrado en el dibujo y borrado de gráficos está oculto al jugador y solo son visibles los cuadros terminados una vez que se han dibujado.

Hay dos maneras de hacer esto en un Spectrum. Un método sólo funcionará en una máquina con 128K, por lo que vamos a dejarlo de lado por el momento. El otro método en la práctica es más complicado pero funcionará en cualquier Spectrum.

Creando un Búfer de Pantalla

La forma más sencilla de implementar el doble buffer en un Spectrum 48K es la creación una pantalla ficticia en otro lugar de la memoria RAM, y dibujar todos los gráficos de fondo y los sprites ahí. Tan pronto como se haya completado nuestra pantalla copiamos esta pantalla ficticia a la pantalla física en la dirección 16384 haciendo:

```
.
.
; código para dibujar todos nuestros sprites etc.
.
.
.
.
; ahora la pantalla se dibuja copiándola a la pantalla física.

       ld hl,49152
       ld de,16384
       ld bc,6912
       ldir
```

Aunque en teoría esto es perfecto, en la práctica copiar 6912 bytes de RAM (o 6144 bytes si ignoramos los atributos de color) de la visualización de la pantalla en cada cuadro es demasiado lento para los juegos de arcade. El secreto consiste en reducir la cantidad de pantalla RAM que se necesita copiar en cada cuadro y encontrar la forma más rápida para transferirla en lugar de la instrucción **LDIR**.

La primera vía consiste en decidir el tamaño de la pantalla de vamos a ver. La mayoría de juegos separan la pantalla en 2 zonas: un panel de estado para mostrar puntuación, vidas y otros elementos de información y una ventana donde se lleva a cabo toda la acción. Como no necesitamos actualizar el panel de estado en cada trama, nuestra pantalla ficticia sólo tiene que ser tan grande como la ventana de acción. Así que si tuvieramos un panel de estado de 80 x 192 pixel en el borde derecho de la pantalla, nos dejaría una ventana de 176x192 píxeles, es decir, nuestra pantalla simulada solamente tendría que ser de 22 caracteres de ancho por 192 píxeles de alto, o 22x192 = 4224 bytes. El desplazamiento manual de 4224 bytes de una parte a otra de la RAM es mucho menos costoso que la manipulación de 6114 bytes. El truco es encontrar un tamaño que sea lo suficientemente grande como para no restringir el juego, y lo suficientemente pequeño para ser manipulado rápidamente. Por supuesto, también es posible que desees hacer el buffer un poco más grande por los bordes. Si bien estos bordes no se muestran en la pantalla son útiles si se quiere recortar sprites a medida que avanzan en la ventana de acción por los lados.

Una vez que hemos establecido definitivamente el tamaño de nuestro búfer, necesitamos escribir una rutina para transferirlo a la pantalla física uno o dos bytes a la vez. Mientras estamos en eso, también puedes volver a ordenar nuestra pantalla intermedia utilizando un método de visualización más lógico que el utilizado por la pantalla física. Podemos hacer concesiones al peculiar ordenamiento de la memoria de pantalla del Spectrum en nuestra rutina de transferencia, es decir, cualquier rutina de gráficos que haga uso de nuestra memoria de pantalla ficticia se pueden simplificar.

Hay dos maneras muy rápidas de mover una pantalla ficticia a la pantalla de visualización. El primer y más sencillo método es el uso de una gran cantidad de instrucciones **LDI** desenrolladas. El segundo y más complicado hace uso de **PUSH** y **POP** para transferir los datos.

Comencemos con **LDI**. Si nuestro buffer es de 22 caracteres de ancho podríamos transferir una sola línea de la memoria intermedia a la pantalla de visualización con 22 instrucciones consecutivas **LDI**, es mucho más rápido usar una gran cantidad de instrucciones **LDI** en lugar de utilizar un único **LDIR**. Podríamos escribir una rutina para transferir nuestros datos a partir de una sola línea a la vez, apuntando con **HL** al comienzo de cada línea de la memoria intermedia, con **DE** a la línea en la pantalla donde hay que ubicarlo, y luego usar 22 instrucciones **LDI** para mover los datos. Sin embargo como cada instrucción **LDI** toma dos bytes de código, es lógico pensar que tal rutina sería al menos de dos veces el tamaño de la memoria intermedia a mover. Un considerable golpe cuando manejamos un poco más de 40K de memoria RAM útil. En su lugar, puede que desees mover las instrucciones **LDI** a una subrutina que copia a la vez una línea de píxeles, o tal vez un grupo de 8 líneas de píxeles. Esta rutina podría entonces ser llamada desde dentro de un bucle, desenrollado o

no, lo que podría hacerse cargo de los registros **HL** y **DE**. (***NdT**: El desenrollado de bucles es un técnica de aceleración usada mucho en código máquina para mejorar la velocidad del programa, consisten en reemplazar el bucle por la repetición del cuerpo del mismo las veces necesarias, de esta manera se eliminan saltos que ralentizan, a cambio de ocupar mas memoria con el programa*).

El segundo método consiste en transferir la pantalla virtual a la real usando instrucciones **PUSH** y **POP**. Si bien esto tiene la ventaja de ser la manera más rápida de hacerlo, hay algunas desventajas. Necesitas control completo del puntero de pila por lo no se puede producir una interrupción a mitad de la rutina. El puntero de pila debe ser almacenado en alguna parte antes de empezar, y hay que restaurarlo inmediatamente después.

La pila del Spectrum se encuentra normalmente por debajo del código de tu programa, pero este método implica el establecimiento de la pila para que apunte a una parte de la memoria intermedia, para a continuación, utilizando **POP**, copiar el contenido de la pantalla ficticia en cada uno de los pares de registro a su vez. El puntero de pila se mueve entonces para apuntar a la RAM en la zona de la pantalla de visualización, antes de que los registros sean empujados a la memoria en orden inverso a aquel en el que fueron introducidos. Es decir, los valores se introducen en la memoria intermedia desde el comienzo de cada línea, y se empujan a la pantalla en el orden inverso, lo que va desde el final de la línea hasta su principio.

A continuación se muestra la parte esencial de la rutina de transferencia de pantalla del juego **Rallybug**. Este utiliza una memoria intermedia de 30 caracteres de ancho, con 28 caracteres visibles en la pantalla. Los restantes 2 caracteres no se muestran de manera que los sprites se mueven lentamente por la pantalla desde el borde, en lugar de aparecer de repente de la nada. Como el ancho de la pantalla visible es de 28 caracteres, esto requiere 14 registros de 16 bits por línea. Obviamente, el Z80A no tiene muchos registros, incluso contando los registros alternativos y los **IX** e **IY**. Por tanto, la rutina del **Rallybug** divide la pantalla en dos mitades de 14 bytes cada una, lo que requiere sólo 7 pares de registros. La rutina establece el puntero de pila al principio de cada línea de la memoria intermedia, para a continuación hacer **POP** de los datos en **AF**, **BC**, **DE** y **HL**. A continuación, intercambia estos registros con el conjunto de registros alternativos con **EXX**, y hace **POP** de 6 bytes más en **BC**, **DE** y **HL**. Estos registros deben ser ahora ser descargados en el área de la pantalla, por lo que el puntero de pila se establece en el punto del final de la línea de la pantalla correspondiente, y **HL**, **DE** y **BC** son "empujados" con **PUSH** a su posición, se restauran los registros alternativos, **HL**, **DE**, **AC** y **AF**, que son respectivamente copiados a su posición. Esto se repite una y otra vez para cada mitad de cada línea de la pantalla, para al final restaurar el puntero de pila a su posición original.

Complicado, si, pero increíblemente rápido.

```
SEG1   equ 16514
SEG2   equ 18434
SEG3   equ 20482

P0     equ 0
P1     equ 256
```

```
P2      equ 512
P3      equ 768
P4      equ 1024
P5      equ 1280
P6      equ 1536
P7      equ 1792

C0      equ 0
C1      equ 32
C2      equ 64
C3      equ 96
C4      equ 128
C5      equ 160
C6      equ 192
C7      equ 224


xfer    ld (stptr),sp       ; guardar puntero de pila.

; Character line 0.

        ld sp,WINDOW        ; inicio del búferde la linea.
        pop af
        pop bc
        pop de
        pop hl
        exx
        pop bc
        pop de
        pop hl
        ld sp,SEG1+C0+P0+14 ; final de la línea de la pantalla.
        push hl
        push de
        push bc
        exx
        push hl
        push de
        push bc
        push af

        .
        .

        ld sp,WINDOW+4784   ; inicio del búfer de la linea.
        pop af
        pop bc
        pop de
        pop hl
        exx
        pop bc
        pop de
        pop hl
        ld sp,SEG3+C7+P7+28 ; final de la línea de la pantalla.
        push hl
        push de
        push bc
        exx
        push hl
        push de
        push bc
        push af
```

```
okay   ld sp,(stptr)       ; restaurar el puntero de pila.
       ret
```

Haciendo scroll en el búfer

Ahora que tenemos nuestra pantalla ficticia, podemos hacer cualquier cosa que nos guste en ella sin riesgo de parpadeo o de otras anomalías en los gráficos, ya que sólo transferimos el contenido a la pantalla física cuando hemos terminado de construir la imagen. Podemos colocar sprites, enmascarados o no, en cualquier lugar que nos guste y en el orden que nos guste. Nosotros podemos movernos alrededor de la pantalla, animar los gráficos de fondo, y lo más importante, ahora podemos hacer scroll en cualquier dirección.

Se requieren diferentes técnicas para diferentes tipos de desplazamiento, aunque todos tienen una cosa en común: como el desplazamiento es una tarea intensiva del procesador, los bucles desenrollados están a la orden del día. El tipo más simple de desplazamiento es un desplazamiento de pixeles individuales a izquierda/derecha. Un scroll a la derecha de un solo píxel nos obliga a establecer en el registro HL el comienzo de la memoria intermedia y, a continuación, ejecutar los dos operandos siguientes una y otra vez hasta llegar a la final del búfer:

```
       rr (hl)             ; rotar bandera de acarreo y 8 bits a la derecha.
       inc hl              ; siguiente dirección del búfer.
```

Del mismo modo, para ejecutar un scroll de un solo píxel hacia la izquierda pondremos en HL el último byte de la memoria y ejecutar estas dos instrucciones hasta llegar al comienzo del búfer:

```
       rl (hl)             ; rotar bandera de acarreo y 8 bits a la izquierda.
       dec hl              ; siguiente dirección del búfer.
```

La mayoría de las veces, sin embargo, podemos hacerlo solo incrementando o disminuyendo el registro **l**, en lugar del par **hl**, acelerando la rutina aún más. Esto tiene el inconveniente de tener que saber exactamente cuando cambiar el byte alto con los cambios de dirección. Por esta razón por lo general fijo mi dirección de búfer permanentemente justo al principio del proyecto, a menudo en la parte superior de RAM, así que no tengo que volver a escribir las rutinas de desplazamiento cuando las cosas cambian de lugar durante el transcurso del proyecto. Al igual que con la rutina para transferir el búfer a la pantalla física, un bucle desenrollado masivo es muy caro en términos de RAM, por lo que es buena idea escribir un bucle desenrollado más pequeño, que desplace por ejemplo 256 bytes a la vez, luego lo llamamos más o menos 20 veces, dependiendo del tamaño del buffer elegido.

Además del scroll de un píxel a la vez, podemos desplazar cuatro píxeles bastante rápidamente también. Mediante la sustitución de **RL (HL)** por **RLD** para el desplazamiento a la izquierda, y **RR (HL)** por **RRD** para el desplazamiento a la derecha, podemos mover 4 píxeles.

El desplazamiento vertical se realiza por desplazamiento de bytes sobre la RAM, de la misma forma que la rutina para transferir la pantalla ficticia hacia la física. Para desplazarse un píxel, fijamos nuestra dirección **DESDE** al inicio de la segunda línea de píxeles y el registro **A** con la dirección de comienzo de la memoria intermedia, a continuación copiamos los datos desde la dirección **DESDE** hacia la dirección **A** hasta llegar al final del búfer. Para desplazarse hacia abajo, tenemos que trabajar en la dirección opuesta, por lo que establecemos nuestro **DESDE** para que apunte al final de la penúltima línea de la memoria intermedia, y ahora **A** apunta a la dirección de nuestra última línea, y trabajamos hacia atrás hasta que se alcancen el inicio del búfer. La ventaja añadida del desplazamiento vertical es que podemos desplazarnos hacia arriba o hacia abajo más de una línea, simplemente alterando las direcciones y la rutina se ejecutará la misma rapidez. En términos generales, no es buena idea desplazarse más de un píxel si tu velocidad de cuadros es inferior a 25 cuadros por segundo, porque parecerá que la pantalla vibra.

Hay otra técnica que puede ser empleada para el desplazamiento vertical, y es la que empleé al escribir el juego **Megablast** para **Your Sinclair** *(**NdT:** Se refiere a la revista inglesa Tu Sinclair)*. Implica el tratamiento de la pantalla intermedia como envuelta sobre si misma. En otras palabras, se utiliza la misma cantidad de RAM para la memoria de pantalla intermedia, pero la parte de la memoria intermedia que se comenzar a copiar a la parte superior de la pantalla puede cambiar de un cuadro al siguiente. Cuando llega al final del búfer, se salta de nuevo al principio. Con este sistema, la rutina para copiar el búfer toma la dirección de inicio de la memoria intermedia de un puntero de 16 bits que podría apuntar a cualquier línea en el búfer, y copia los datos en la pantalla física línea por la línea hasta que llega al final de la memoria intermedia. En este punto, la rutina copia los datos desde el principio de la memoria intermedia hacia el resto de la pantalla física. Esto hace que la transferencia de la rutina sea un poco más lenta, y complica cualquier otra rutina de gráficos, que también tienen que volver a la primera línea cada vez que llega a la última línea del búfer. Hacerlo, por otro lado, significa que no hay necesidad de desplazar datos para hacer scrool vertical de la pantalla. Al cambiar el puntero de 16 bits con la primera línea que se copia a la pantalla física, el desplazamiento se realiza automáticamente cuando el búfer se transfiere.

Capítulo 14: Movimiento sofisticado

Hasta ahora hemos movido sprites arriba, abajo, a la izquierda y a la derecha por píxeles. Sin embargo, muchos juegos requieren una manipulación de sprites más sofisticada. Los juegos de plataformas requieren manejar la gravedad, los juegos de carreras top-down utilizan el movimiento de rotación, otros juegos utilizan la inercia.

Tablas de salto y de inercia

La forma más sencilla de manejar la gravedad o la inercia es disponer de una tabla de valores. Por ejemplo, los **Egghead games** hacen uso de una tabla de saltos y mantienen un puntero a la posición actual. Dicha tabla puede tener un aspecto como la siguiente.

```
; Tabla de saltos.
; Valores mayores de 128 son de subida, menores de 128 de bajada.
; Conforme el valor se aleja de 128 se aumenta la velocidad.

jptr   defw jtabu
jtabu  defb 250,251,252
       defb 253,254,254
       defb 255,255,255
       defb 0,255,0
jtabd  defb 1,0,1,1,1,2
       defb 2,3,4,5,6,6
       defb 6,6,128
```

Con el puntero almacenado en **jptr**, podríamos hacer algo como esto:

```
       ld hl,(jptr)        ; recuperar el puntero del salto.
       ld a,(hl)           ; siguiente valor.
       cp 128              ; ¿alcanzado el final de la tabla?
       jr nz,skip          ; no, estamos bien.
       dec hl              ; atrás aumentamos la velocidad.
       ld a,(hl)           ; recuperar la velocidad máxima.
skip   inc hl              ; avanzar el puntero.
       ld (jptr),hl        ; establecer la siguiente posición del puntero.
       ld hl,verpos        ; posición vertical del jugador.
       add a,(hl)          ; agregar la cantidad correspondiente.
       ld (hl),a           ; Establecer la nueva posición del jugador.
```

Para iniciar un salto, debemos establecer **jptr** a **jtabu**. Para empezar a caer, hay que configurarlo con **jtabd**.

Bien, esta descripción es muy simple. En la práctica, debemos utilizar el valor de la tabla de saltos como un contador de bucles, mover al jugador arriba o hacia abajo un píxel cada vez, comprobar colisiones con plataformas, paredes, elementos mortales, etc. como siempre. También podríamos utilizar el elemento final (128) para indicar que el jugador ha caído desde demasiado lejos, y

establecer un indicador para que la próxima vez que el jugador golpee contra algo sólido pierda una vida. Considerando esto, ya tienes tu rutina.

Coordenadas fraccionarias

Si queremos manejar de forma más sofisticada la gravedad, la inercia o el movimiento de rotación, necesitamos coordenadas fraccionarias. Hasta ahora, con la resolución del Spectrum de 256x192 píxeles, sólo hemos tenido que utilizar un byte por coordenadas. Si por el contrario utilizamos un par de registros de dos bytes, el byte alto para la parte entera y el byte bajo para la fraccionaria, se abre un nuevo mundo de posibilidades. Esto nos da 8 bits para cifras decimales binarias, lo que permite movimientos muy precisos y sutiles. Con una coordenada en el par **hl**, podemos configurar el desplazamiento en **de**, y unir los dos. Al presentar nuestros sprites, simplemente usamos los bytes altos como nuestra coordenadas X e Y para nuestro cálculo de la dirección de la pantalla, y desechamos los bytes inferiores que contienen las fracciones. El efecto de sumar una fracción a una coordenada no será visible cada cuadro, pero incluso la fracción más pequeña, 1/256, moverá lentamente un sprite con el tiempo.

Ahora podemos echar un vistazo a la gravedad. Esta es una fuerza constante, en la práctica acelera un objeto hacia el suelo a 9,8 m/s^2. Para simularlo en un juego de Spectrum, haremos que nuestra coordenada vertical sea una palabra de 16 bits. A continuación, establecemos una segunda palabra de 16 bits para la inercia. Cada cuadro añadimos una pequeña fracción a la inercia, y a continuación añadimos la inercia a la posición vertical. Por ejemplo:

```
       ld hl,(vermom)      ; inercia.
       ld de,2             ; fracción pequeña, 1/128.
       add hl,de           ; incrementar inercia.
       ld (vermom),hl      ; guardar inercia.
       ld de,(verpos)      ; posición vertical.
       add hl,de           ; añadir inercia.
       ld (verpos),hl      ; guardar nueva posición.
       ret
verpos defw 0              ; posición vertical.
vermom defw 0              ; inercia vertical.
```

A continuación, para mostrar nuestros sprites, simplemente tomamos el byte alto de nuestra posición vertical, **verpos+1**, que nos dan el número de píxeles desde la parte superior de la pantalla. Diferentes valores de **de** variarán la fuerza de la gravedad, de hecho, incluso podemos cambiar la dirección restando **de** con **hl**, o añadiendo una distancia negativa (65536-distancia). Podemos aplicar lo misma también a la coordenada **y** para mantener el sprite sujeto a la inercia en todas direcciones. Así es como nos gustaría ir escribiendo un juego de estilo similar a **Thrust**.

Movimiento rotatorio

Otro tema que podemos necesitar para juegos del estilo al **Thrust**, carreras top-down, u otros en los que círculos o trigonometría básica están involucrados es una tabla de senos y cosenos. Las

matemáticas no son del agrado de todo el mundo, y si tu trigonometría está un poco oxidada te sugiero leer sobre senos y cosenos antes de continuar con el resto del capítulo.

En matemáticas, podemos encontrar la distancia x e y desde el centro de un círculo dado el radio y el ángulo mediante el uso de senos y cosenos. Sin embargo, mientras que en matemáticas un círculo se compone de 360 grados o lo que es lo mismo de 2 PI radianes, es más conveniente para el programador de Spectrum representar un ángulo como, por ejemplo, un valor de 8 bits de 0 a 255, o incluso usar menos bits, dependiendo del número de posiciones que el sprite del jugador pueda tomar. Podemos utilizar este valor para mirar su valor fraccionario de 16 bits para el seno y el coseno de una tabla. Suponiendo que tenemos un ángulo de 8 bits almacenado en el acumulador y deseamos encontrar el seno, simplemente accedemos a la tabla de manera similar a la siguiente:

```
       ld de,2             ; fracción pequeña - 1/128.
       ld l,a              ; ángulo en el byte bajo.
       ld h,0              ; ponemos a cero byte alto.
       add hl,hl           ; doble desplazamiento las entradas son de 16 bits.
       ld de,sintab        ; dirección de tabla de senos.
       add hl,de           ; añadir el desplazamiento de este ángulo.
       ld e,(hl)           ; fracción del seno.
       inc hl              ; apunto a la segunda mitad.
       ld d,(hl)           ; parte entera.
       ret                 ; volver con el seno en DE.
```

Realmente el BASIC de Sinclair ya nos proporciona los valores que necesitamos, con sus funciones **SIN** y **COS**. Usándolas, podemos almacenar con **POKE** en la memoria RAM, o bien guardarlos en cinta, o guardarlos en binario utilizando un emulador como **SPIN**. Alternativamente, es posible que prefieras utilizar otro lenguaje de programación en el PC para generar una tabla formateada de valores de senos, para importarlos en tu archivo de origen o incluirlos como valores binarios. Para una tabla de senos con 256 ángulos equidistantes se necesitaría un total de 512 bytes, pero debemos que tener cuidado para convertir el número devuelto por **SIN** en uno que nuestro juego reconozca. Multiplicando el seno por 256 nos dará nuestros valores positivos, pero cuando **SIN** devuelve un resultado negativo, podrías tener que multiplicar el valor absoluto del seno por 256, y luego restarlo de 65536 o poner el bit d7 del byte a uno para indicar que el número debe ser restado en lugar de sumado a nuestras coordenadas. Con una tabla de senos construida de esta manera no necesitamos una tabla separada para los cosenos, solo hemos de añadir o restar 64, o un cuarto de vuelta, al ángulo antes de obtener el valor en nuestra tabla. Para mover un sprite en un ángulo de A, se añade el seno de A a una coordenadas, y el coseno de A a la otra coordenada. Cambiando si sumamos o restamos un cuarto de vuelta para obtener el coseno, y viendo que cuadrante utiliza senos y cual cosenos, podemos empezar nuestro círculo en cualquiera de los 4 puntos cardinales y hacer que se mueva en sentido horario o anti horario.

Capítulo 15: Matemáticas

Sumar y restar es bastante sencillo para la CPU del Spectrum, tenemos una gran cantidad de instrucciones para llevar a cabo estas tareas. Pero a diferencia de algunos procesadores posteriores en su misma serie, el programador del Z80A tiene que hacer por si mismo la multiplicación y la división. Aunque estos cálculos son poco frecuentes, tienen sus usos en ciertos tipos de juego y hasta que tengas rutinas para hacerlo, ciertas cosas son difíciles de hacer. Por ejemplo, sin el teorema de Pitágoras puede ser difícil programar para que un sprite enemigo dispare al jugador con cierto grado de precisión.

Supongamos que un sprite **A** necesita hacer un disparo al sprite **B**. Tenemos que encontrar el ángulo en el que el sprite **A** debe disparar y necesitamos algo de trigonometría para hacerlo. Sabemos las coordenadas de los sprites, **Ax**, **Ay**, **Bx** y **By**, y las distancias entre estos, **Bx-Ax** y **By-Ay**, eso nos darán la longitud de los catetos opuesto y contiguo. Por desgracia, la única manera de calcular el ángulo entre opuesto y contiguo es el uso del arco tangente, y las tangentes solo son adecuadas para ciertos ángulos, por lo que es mejor usar senos o cosenos en su lugar. Así que con el fin de encontrar el ángulo entre el sprite **A** y el sprite **B**, tenemos que encontrar la longitud de la hipotenusa.

La hipotenusa se calcula elevando al cuadrado la distancia en **x**, añadiéndola al cuadrado de la distancia en **y**, para a continuación encontrar la raíz cuadrada de la suma. Existen rutinas en la ROM de Sinclair para hacer todo esto, pero hay un serio inconveniente: como puede decirte cualquier persona que haya usado Sinclair BASIC alguna vez, las rutinas matemáticas son increíblemente lentas como para usarlas en la escritura de juegos. Así que tenemos que poner a trabajar los dedos y escribir nuestras propias rutinas.

Los cuadrados de nuestras distancias **x** e **y** se consiguen usando una rutina de multiplicación, multiplicando los números por sí mismos. Afortunadamente, esta parte es relativamente indolora. La multiplicación se logra de la misma manera como se haría la multiplicación larga en papel, aunque esta vez estamos trabajando en binario. Todo lo que se requiere es rotaciones, pruebas de bits y sumas. Cuando un bit exista en nuestro primer factor, añadimos el segundo factor al total. Luego desplazamos el segundo factor a la izquierda, y probamos el siguiente bit de nuestro primer factor. La rutina de abajo, tomada del juego **Kuiper Persecution**, demuestra esta técnica, multiplica **h** por **d** y devuelve el resultado en **hl**.

```
imul   ld e,d              ; HL = H * D
       ld a,h              ; poner en el acumulador el primer multiplicando.
       ld hl,0             ; poner a cero el resultado.
       ld d,h              ; poner a cero el byte alto ya que de=multiplicador.
       ld b,8              ; repetir 8 veces.
imul1  rra                 ; rotar el bit mas a la derecha en el acarreo.
       jr nc,imul2         ; no se estableció.
       add hl,de           ; se estableció el bit, por lo que añadir a de.
       and a               ; restablecer acarreo.
imul2  rl e                ; aumentar 1 bit izquierdo.
       rl d
       djnz imul1          ; repetir 8 veces.
```

```
        ret
```

Ahora necesitamos una raíz cuadrada, que es donde empiezan los problemas. Las raíces cuadradas son mucho más complicadas. Esto significa hacer una gran cantidad de divisiones, por lo que primero necesitamos una rutina de división. Esto se puede ver como trabajar de manera opuesta a la multiplicación, desplazando y restando. La siguiente rutina, también del juego **Kuiper Persecución**, divide **hl** por **d** y devuelve el resultado en **hl**.

```
idiv   ld b,8             ; bits a comprobar.
       ld a,d             ; número por el que dividir.
idiv3  rla                ; comprobar bit de la izquierda.
       jr c,idiv2         ; no requiere más desplazamientos.
       inc b              ; necesita desplazamientos extra.
       cp h
       jr nc,idiv2
       jp idiv3           ; repetir.

idiv2  xor a
       ld e,a
       ld c,a             ; resultado.
idiv1  sbc hl,de          ; restarlo.
       jr nc,idiv0        ; no hay acarreo, mantener el resultado.
       add hl,de          ; restaurar el valor original de hl.
idiv0  ccf                ; invertir bit de acarreo.
       rl c               ; rotar en ac.
       rla
       rr d               ; dividir por 2.
       rr e
       djnz idiv1         ; repetir.
       ld h,a             ; copiar el resultado a hl.
       ld l,c
       ret
```

De la misma manera que la multiplicación se compone de desplazamiento y suma, y la división se realiza mediante desplazamiento y resta, las raíces cuadradas se pueden calcular por desplazamiento y división. Simplemente estamos tratando de encontrar el número que "mejor se ajuste", de forma que multiplicado por sí mismo nos da el número con el que empezamos. No voy a entrar en la explicación detallada de cómo trabaja la siguiente rutina, si estás muy interesado sigue mis comentarios y haz una ejecución paso a paso con un depurador. Tomado del juego **Blizzard's Rift**, devuelve la raíz cuadrada de hl en el acumulador.

```
isqr   ld (sqbuf0),hl     ;  número del que queremos encontrar la raíz
cuadrada.
       xor a              ; poner a cero el acumulador.
       ld (sqbuf2),a      ; buffer resultado.
       ld a,128           ; comenzar con la división con el bit alto.
       ld (sqbuf1),a      ; siguiente divisor.
       ld b,8             ; 8 bits a dividir.
isqr1  push bc            ; guardar contador de bucles.
       ld a,(sqbuf2)      ; resultado actual.
       ld d,a
```

```
        ld a,(sqbuf1)       ; siguiente bit a comprobar.
        or d                ; combinar con divisor.
        ld d,a              ; almacenar byte bajo.
        xor a               ; HL = HL / D
        ld c,a              ; poner a cero c.
        ld e,a              ; poner a cero e.
        push de             ; recordar divisor.
        ld hl,(sqbuf0)      ; número original.
        call idiv4          ; dividir el número por d.
        pop de              ; restaurar divisor.
        cp d                ; ¿el divisor es mayor que el resultado?
        jr c,isqr0          ; sí, no guardar este bit entonces.
        ld a,d
        ld (sqbuf2),a       ; guardar nuevo divisor.
isqr0   ld hl,sqbuf1        ; bit que comprobamos.
        and a               ; limpiar bandera de acarreo.
        rr (hl)             ; siguiente bit a la derecha.
        pop bc              ; restaurar el contador del bucle.
        djnz isqr1          ; repetir.
        ld a,(sqbuf2)       ; retornar resultado en hl.
        ret
sqbuf0 defw 0
sqbuf1 defb 0
sqbuf2 defb 0
```

Con la longitud de la hipotenusa calculada, podemos dividir simplemente el cateto opuesto por la hipotenusa para encontrar el coseno del ángulo. Una búsqueda rápida en nuestra tabla de senos nos dará el ángulo. ¡Uf!

Este es todo el cálculo tomado del juego **Blizzard's Rift**. Debes notar que usa la longitud del cateto contiguo en lugar del opuesto, por lo que se encuentra el arco coseno en lugar del arco seno. Solo se utiliza cuando la nave está por encima del cañón de la torreta, dando al jugador la oportunidad de acercarse sigilosamente y atacar desde abajo. Sin embargo demuestra cómo un sprite puede disparar a otros con precisión mortal. Si has jugado alguna vez al **Blizzard's Rift**, sabrás exactamente lo letales que pueden ser los cañones de las torretas.

```
; La nave debe estar por encima del cañón para que podamos emplear trigonometría
; básica para apuntar.
; Tenemos que encontrar el ángulo, para ello dividimos el cateto contiguo por
; la hipotenusa y encontramos el arco coseno.

; En primer lugar ponemos la distancia al oponente en la pila:

mgunx   ld a,(nshipy)       ; coordenada y de la nave.
        ld hl,guny          ; coordenada y del cañón.
        sub (hl)            ; encontrar la diferencia.
        jr nc,mgun0         ; el resultado fue positivo.
        neg                 ; negativo, hacerlo positivo.
mgun0   cp 5                ; ¿diferencia y menor de 5?
        jr c,mgunu          ; sí, apuntar hacia arriba.
        push af             ; colocar longitud del oponente en la pila.

; A continuación se necesita la longitud de la hipotenusa para lo que
; podemos usar el viejo teorema de Pitágoras.
```

```
       ld h,a               ; copiar a en h.
       ld d,h               ; copiar h en d.
       call imul            ; multiplicar parte entera obtener resultado 16 bits.
       push hl              ; guardar el valor al cuadrado.

       ld hl,nshipx         ; cañón coordenada x.
       ld a,(gunx)          ; nave coordenada x.
       sub (hl)             ; encontrar la diferencia, siempre será positiva.
       ld h,a               ; poner la diferencia de x en h.
       ld d,h               ; copiar h en d.
       call imul            ; multiplicar h por d para obtener el cuadrado.

       pop de               ; obtener el último resultado al cuadrado.
       add hl,de            ; necesitamos la suma de los dos.
       call isqr            ; encontrar la raíz cuadrada, hipotenusa en a.
       pop de               ; cateto opuesto ahora en el registro d..

       ld h,a               ; longitud de la hipotenusa.
       ld l,0               ; ninguna fracción o signo.
       ex de,hl             ; intercambiarlos.

; Cateto opuesto e hipotenusa se encuentran ahora en de y hl.
; Ahora dividimos la primera por la segunda y encontramos el arco seno.
; Recuerda: seno = cateto opuesto sobre la hipotenusa.

       call div             ; la división nos dará el seno.
       ex de,hl             ; queremos el resultado en de.
       call asn             ; obtener arco seno para encontrar el ángulo.
       push af

; Bien, tenemos el ángulo, pero está sólo entre 0 y PI/2 radianes (64 ángulos)
; por lo que tenemos que hacer un ajuste en base al cuadrante del círculo.
; Podemos establecer en qué cuadrante del círculo está nuestro ángulo
; examinando la diferencia entre las coordenadas de la nave y el arma.

       ld a,(guny)          ; cañón posición y.
       ld hl,shipy          ; nave y.
       cp (hl)              ; ¿está la nave a la derecha?
       jr nc,mgun2          ; jugador a la izquierda, ángulo en segundo
cuadrante.

; Ángulo del jugador en el primer cuadrante, por lo que debemos restar de 64.

       ld a,64              ; pi/2 radianes = 64 ángulos.
       pop bc               ; ángulo en b.
       sub b                ; restarlos.
       ld (ix+1),a          ; nuevo ángulo.
       ret                  ; tenemos nuestro ángulo.

; Segundo cuadrante, añadir el literal 64 a nuestro ángulo.

mgun2  pop af               ; ángulo original.
       add a,192            ; sumar pi/2 radianes.
       ld (ix+1),a          ; nuevo ángulo.
       ret                  ; ¡buen trabajo!
```

Capítulo 16: Música y efectos AY

***NdT**: Mira esta entrada que hice sobre el sonido en 8 bits que puede ayudarte a enter esta parte.*

El AY-3-8912

Introducido con los modelos 128K y prácticamente estándar desde entonces, fue un chip de sonido muy popular que se utilizaba en otros muchos equipos, por no hablar de vídeo juegos y máquinas de pinball. Básicamente dispone de 14 registros que se pueden leer o escribir a través de instrucciones **in** y **out**.

Los primeros seis registros controlan el tono para cada uno de los tres canales, son parejas en modo little-endian (***NdT**: primero el byte bajo*) como es de esperar, es decir, el **registro 0** es para byte bajo para el tono del canal A, el **registro 1** es el byte alto del tono del canal A, el **registro 2** es el byte bajo para el tono del canal B, y así sucesivamente. El **registro 6** controla el periodo del ruido blanco, con valores válidos de 0 a 31. 0 significa ruido de las frecuencias más alta, 31 de las más bajas. El **registro 7** es el control del mezclador. Los bits D0-D5 seleccionar el ruido blanco, el tono, ninguno o ambos. Para activar un tono o el ruido blanco, se debe establecer un bit, siendo 0 salida de los tres canales de tono mas el ruido blanco, y 63 no pone nada a la salida. Los **registros 8, 9 y 10** son para controles de amplitud y envolvente. Un valor de 16 indica al chip utilizar el generador de envolventes, y 0-15 ajusta el volumen de ese canal directamente. En la práctica, es mejor controlar el volumen por ti mismo junto con el tono. Mediante la variación de estos de un cuadro al siguiente es posible producir una variedad muy buena de efectos de sonido. Si deseas utilizar el generador de envolventes, los dos siguientes, **registros 11 y 12**, se emparejan para formar el período de 16 bits y el **registro 13** determina el patrón del envolvente. El manual del 128K explica la lista completa de los patrones, pero no lo cubro aquí ya que nunca los encontré particularmente útiles.

Para leer un registro, se escribe el número del registro en el puerto 65533, a continuación se lee ese puerto. Para escribir en un registro, se envía de nuevo el número del registro al puerto 65533, y luego el valor al puerto 49149. Para los no iniciados, los opcodes del Z80 no parecen ser capaces de escribir direcciones de puerto de 16 bits. No dejes que te confunda, es sólo que la forma en que se escriben es engañosa. En realidad **out (c),a** realiza **out (bc),a**, y **out (n),a** en realidad hace **out (a*256+n),a**.

La lectura de un registro del chip de sonido tiene varios usos. Es posible que desees leer los registros de volumen 8, 9 y 10 y presentar una barra de volumen. Hice algo similar en **Egghead 5** Además, aunque no lo creas, la pistola de Sinclair se lee a través del registro 14 del chip de sonido. Sí, de verdad. En realidad solo genera dos informaciones, si se presiona el gatillo o no, y si el arma está apuntando a una parte brillante de la pantalla (o de hecho a cualquier objeto brillante). Es responsabilidad del programador lo que hace con esa información.

Aquí algo de código básico para escribir en el chip de sonido:

```
;Escribir el contenido de nuestro buffer AY en los registros el AY.

w8912  ld hl,snddat        ; inicio de los registros del AY-3-8912.
       ld e,0              ; comenzar con el registro 0.
       ld d,14             ; escribir 14.
       ld c,253            ; byte bajo del puerto a escribir.
w8912a ld b,255            ; 255*256+253 = puerto 65533 = seleccionar registro.
       out (c),e           ; indicar al chip en que registro estamos
escribiendo.
       ld a,(hl)           ; valor a escribir.
       ld b,191            ; 191*256+253 = puerto 49149 = escribir en registro.
       out (c),a           ; esto es lo que estamos escribiendo allí.
       inc e               ; siguiente registro del chip de sonido.
       inc hl              ; siguiente byte a escribir.
       dec d               ; decrementar contador de bucle.
       jp nz,w8912a        ; repetir hasta completar.
       ret

snddat defw 0              ; registro de tono, canal A.
       defw 0              ; registro de tono, el canal B.
       defw 0              ; igual para el canal C.
sndwnp defb 0              ; período del ruido blanco.
sndmix defb 60             ; Tono/ruido control del mezclador.
sndv1  defb 0              ; canal A generador de amplitud/envolvente.
sndv2  defb 0              ; canal B amplitud/envolvente.
sndv3  defb 0              ; canal C amplitud/envolvente.
sndenv defw 600            ; duración de cada nota.
       defb 0
```

Llamando a **w8912** en cada iteración del bucle principal el sonido es actualizado constantemente. Tu solo debes actualizar la memoria intermedia, ya que el sonido cambia de un cuadro al siguiente. Piensa en ello como en un "animador" del sonido. Sin embargo, sólo porque dejes de actualizar el registro de sonido este no va a dejar de sonar. El chip AY mantendrá sonando tu tono o ruido hasta que le indiques que se detenga. Una forma rápida de hacer esto es establecer los tres registros de amplitud a 0. En el ejemplo anterior, escribir un cero en **sndv1**, **sndv2** y **sndv3** y luego llamar **w8912**.

Usando drivers para la Música

La mayoría de los driver de música de los 128K (y de algunos 48K), tienen dos puntos de entrada. Una rutina de inicialización/terminación que detiene todo sonido y restablece el driver al comienzo de la melodía, y una rutina de servicio para ser llamada varias veces, por lo general 50 veces por segundo. Un buen lugar para almacenar música suele ser 49152, el inicio del banco de RAM conmutable. Si conoces la dirección de inicio del controlador, o puedes determinarla por ti mismo, el principio general es el de inicializar la dirección en la que se carga la música. La mayoría de las veces el código en este punto es simplemente saltar a otra dirección, o bien carga un registro o un par de registros antes de saltar a otra parte. La rutina de servicio tiende a realizar inmediatamente este **jp**. Si el driver no tiene otra documentación, es posible tener que desmontar el código para encontrar esta dirección, por lo general en 3-6 Bytes.

Para utilizar un driver de música, llamar a la dirección de inicialización antes de comenzar y cuando quieras apagarla. Entre esos puntos es necesario llamar a la rutina de servicio en varias ocasiones. Esto puede hacerse o bien manualmente, o mediante el establecimiento de una interrupción para hacer el trabajo de forma automática. Si decides hacerlo de forma manual, por ejemplo en el código de menú, ten en cuenta que la limpieza de la pantalla, mostrar el menú, la tabla de puntuación mas alta, las instrucciones, etc. tardarán más de 1/50 de segundo, así que esto retrasará su rutina y podría sonar de forma extraña. Podría ser mejor escribir una rutina para borrar la pantalla durante varios cuadros con algún tipo de efecto especial, marcar cuando se detiene y llama a la rutina de servicio en cada cuadro.

Capítulo 17: Interrupciones

La creación de tus propias interrupciones puede ser una pesadilla la primera vez que lo intentes, ya que es un asunto complicado. Con práctica se convierte en un poco más fácil. Para hacer que el Spectrum ejecute nuestra propia rutina de interrupción hay que decirle en que ubicación está nuestra rutina, poner la máquina en **modo 2** de interrupción, y garantizar que las interrupciones están habilitadas. ¿Suena bastante simple? La parte difícil es decirle al Spectrum, donde se encuentra nuestra rutina.

Con la máquina en **modo 2**, el Z80 utiliza el registro **i** para determinar el byte alto de la dirección del puntero a la dirección de la rutina de servicio de interrupción. El byte bajo es suministrado por el hardware. En la práctica, nunca se sabe lo que va a contener el byte bajo, ¿ves el problema? El byte bajo igual podría ser 0, que podría ser 255, o que podría estar en cualquier posición intermedia. Esto significa que necesitamos todo un bloque de 257 bytes de punteros a la dirección de inicio de nuestra rutina de servicio. Como el byte bajo suministrado por el hardware puede ser par o impar, tenemos que asegurarnos de que el byte bajo y el byte alto de la dirección de nuestra rutina de servicio son idénticos. Esto limita seriamente donde podemos localizar nuestra rutina. También debemos localizar nuestra tabla de punteros y nuestra rutina sólo en la memoria RAM no contenida. No lo coloques por debajo de la dirección 32768. Incluso ubicándolo en la memoria RAM no contenida, como por ejemplo en el banco 1, producirá problemas en ciertos modelos del Spectrum. Personalmente, encuentro el banco 0 un lugar tan bueno como cualquier otro.

Digamos que elegimos la dirección 51400 como la ubicación de nuestra rutina de interrupción. Esto es válido ya que tanto el byte alto como el byte bajo son 200, ya 200*256+200=51400. A continuación, necesitamos una tabla de 129 punteros que apunten todos a esta dirección, o 257 veces **DEFB 200**, situada en el inicio de un límite de página de 256 bytes. Suponiendo que lo ponemos en lo alto del camino, podríamos empezar por 254*256=65024.

Debemos hacer esto

```
       org 51400

int    ; rutina de servicio de interrupción.


       org 65024

       ; punteros a rutina de interrupción.

       defb 200,200,200,200
       defb 200,200,200,200
       .
       .
       defb 200,200,200,200
       defb 200
```

¡Uf! Aún así, ahora llegamos a nuestra rutina de interrupción. Las interrupciones pueden ocurrir durante cualquier período, por lo que tenemos que conservar los registros que es probable que se utilicen, ejecutar nuestro código, llamar opcionalmente a la rutina de servicio de la ROM, restaurar los registros, volver a habilitar las interrupciones y por último regresar de la interrupción con un **reti**. Nuestra rutina podría asemejarse a esta:

```
int     push af             ; preservar registros.
        push bc
        push hl
        push de
        push ix
        call 49158          ; hacer sonar la música.
        rst 56              ; rutina ROM, leer teclas y actualizar el reloj.
        pop ix              ; restaurar registros.
        pop de
        pop hl
        pop bc
        pop af
        ei                  ; Volver a habilitar interrupciones antes de 
regresar.
        reti                ; echo.
        ret
```

Si no estás leyendo el teclado a través de las variables del sistema, puedes desear prescindir del **rst 56**. Si lo haces, vas a liberar los registros **iy**. Sin embargo, si el temporizador de tu juego cuenta los cuadros usando el método descrito en el capitulo de temporización, tendrás que incrementar el contador de cuadros por ti mismo:

```
        ld hl,23672         ; contador de cuadros.
        inc (hl)            ; aumentarlo.
```

Con todo esto en su lugar, estamos listos para deshabilitar las interrupciones. Hay que apuntar al registro **i** en la tabla de punteros y seleccionar el **modo 2** de interrupción. Este código va a hacer el trabajo por nosotros:

```
        di                  ; deshabilitar las interrupciones por precaución.
        ld a,200            ; byte alto del puntero a la tabla de ubicaciones.
        ld i,a              ; establecer el byte alto.
        im2                 ; seleccionar el modo2 de interrupción.
        ei                  ; habilitar las interrupciones.
```

Capítulo 18: Juegos que carguen y se ejecuten automáticamente

Aunque esto es bastante simple para un programador experimentado en Sinclair BASIC, se trata de un área que se suele pasar por alto. En particular, los programadores que se han pasado al Spectrum desde otras máquinas no estarán familiarizados con la forma como se hace.

Para poder ejecutar una rutina en código de máquina tenemos que empezar desde el BASIC. Usando este escribimos un pequeño programa cargador básico, que reserva sitio para el código máquina, carga dicho código y luego lo ejecuta. El tipo más simple de cargador sería el contenido es estas líneas:

```
10 CLEAR 24575: LOAD ""CODE: RANDOMIZE USR 24576
```

El primer comando, **CLEAR**, establece la variable **RAMTOP** por debajo de la zona ocupada por el código máquina, de modo que el BASIC no lo sobrescriba. También borra la pantalla y mueve la pila fuera del camino. El número que sigue por lo general debe estar un byte por debajo del primer byte de tu juego. **LOAD ""CODE** carga el siguiente archivo de código desde la cinta, y **RANDOMIZE USR** llama a la rutina en código máquina en la dirección especificada, en este caso 24576. Este debe ser el punto de entrada para tu juego. En un Spectrum, la ROM se encuentra en los primeros 16K, y va seguido por varias cosas como la memoria de video, las variables del sistema y del BASIC. Un lugar seguro para tu código es por encima de esta zona, en cualquier lugar hasta el final de la RAM en la dirección 65535. Solo con un pequeño cargador en BASIC a la dirección de inicio 24576, o incluso 24000, te dará un montón de espacio para su juego.

Este programa cargador se guarda a continuación en cinta con un comando como este

```
SAVE "name" LINE 10
```

LINE 10 indica que cuando termine de cargar, el programa en BASIC se auto-ejecuta desde la línea 10.

Después del cargador BASIC viene el archivo de código. Puedes guardar un archivo de código así:

```
SAVE "name" CODE 24576,40960
```

CODE le dice al Spectrum que guarde un archivo de código, en lugar de en BASIC. El primer número de después es la dirección de inicio del bloque de código, y el último número es su longitud.

Esto es bastante simple, pero ¿y si queremos añadir una pantalla de carga? Bueno, es bastante sencillo. Podemos cargar una pantalla utilizando

```
LOAD ""SCREEN$
```

Lo que esto va a hacer es cargar un bloque de código de 6912 bytes de longitud, al inicio de la dirección de pantalla en 16384. Poner el archivo de pantalla en memoria es un poco más complicado, ya que no podemos simplemente guardar la pantalla como un archivo ya que las dos líneas de la parte inferior se pueden sobrescribir con mensajes del tipo "***Start tape, then press any key***". Así que cargaremos nuestra imagen en otro punto de la memoria RAM, como 32768, para a continuación utilizar

```
SAVE "name" CODE 32768,6912
```

6912 es el tamaño de la RAM de pantalla del Spectrum. Cuando volvemos a cargar el bloque de cinta utilizando **LOAD ""SCREEN$**, estamos especificando que queremos forzar a que el archivo de código se cargue en la memoria de pantalla. En estas circunstancias, no importa donde se encontraba el archivo de código cuando se guardó.

Ahora tenemos otro posible problema: ¿el mensaje **Bytes: name** que se imprime antes de la carga del código no aparecerá tapando parte de nuestra pantalla? Bueno, sí que lo hará. Podemos superar esto pokeando en el flujo de salida.

```
POKE 23739,111
```

Esto hará el truco para nosotros. Así que nuestro cargador BASIC ahora se ve así:

```
10 CLEAR 24575: LOAD ""SCREEN$: POKE 23739,111: LOAD ""CODE: RANDOMIZE USR 24576
```

Capítulo 19: Diseño de juegos

Cincuenta por cien Arte, Cincuenta por cien Ciencia

Ahora ya debes tener suficientes conocimientos para ser capaz de juntar las instrucciones para formar el motor de un juego. Felicitaciones, ahora conoces el cincuenta por ciento de la manera de escribir un juego. Este capítulo tiene como objetivo hacer frente a la otra mitad. Es fácil caer en la trampa de pensar que la escritura de un juego es tan simple como aprender cómo ordenar una enorme variedad de instrucciones en una secuencia coherente, con el fin de llevar a cabo un gran plan. El aspecto técnico es, por supuesto, esencial. Sin embargo, un buen desarrollador de juegos tiene que ser mucho más que un técnico especializado. Tiene que ser también un artista. Eso no significa que debas ser un artista de los gráficos, aunque estos deben ser agradables visualmente no es a lo que me refiero. Estoy, por supuesto, hablando sobre el arte del diseño de juegos.

Se podría optar por seguir el camino de la técnica, escribit tu código para deslumbrar a otros "**techies**" que saben cómo funciona el Spectrum y lo que puede hacer normalmente. Sin embargo eso es poco probable que impresione a todo el mundo. Para empezar, el juego puede parecer increíble pero podría ser aburrido de jugar y no mantener la atención más de cinco minutos. Lo que es más, no todos los fans del Spectrum conoce las limitaciones de la máquina. Si realmente deseas maximizar tu audiencia, necesitas apelar a los que no saben ni le importa lo que el Spectrum puede hacer. Considera esto: imagina que dos programadores de Commodore 64 liberan sus juegos el mismo día. Uno de ellos escribió un clon del ***Space Invaders***, que sorprendió a los chicos mas técnicos de la escena de Commodore, y el otro escribió un juego técnicamente nada especial pero adictivo, diferente a todo lo que se ha visto antes. Como propietario de un Spectrum que tienes poco conocimiento de los límites de hardware del C64, ¿a cual sería más probable que echaras un vistazo? Los clones del ***Space Invaders*** pueden mostrar una increíble magia técnica para esa máquina, pero ¿no es probable que no veas nada que no hayas visto antes en algún otro formato? Consideremos ahora ese escenario al revés: ¿por qué un fan de Commodore, Amstrad, Acorn u Oric quiere descargar tu juego? ¿Que lo hace tan especial que no puedes encontrar algo similar o mejor en su máquina favorita?

Si deseas maximizar el interés, es buena idea ofrecer algo único.

Mecánicas de juegos

Escribir el código del programa es como hacer una gran construcción de Lego formada por miles de minúsculos ladrillos, las instrucciones individuales que dicen a la CPU qué hacer. La mecánica del juego es más afin a Duplo; piezas más grandes pero aún con infinitas maneras de reorganizarlas para alcanzar el efecto que deseas. Un enfoque para el diseño de juegos podría ser considerar los siguientes pasos.

En primer lugar, empieza por considerar el método de control del jugador. ¿Cómo se mueve? Piensa si se limita de alguna manera, y si es así, ¿cómo? ¿Será capaz de realizar acciones o completar

tareas para aumentar o modificar su capacidad de maniobra de alguna manera? En líneas generales, cualquier método de control que puedas concebir probablemente ya se ha hecho antes, pero eso no debe impedir que te hagas estas preguntas y busques interesantes variaciones.

En segundo lugar, ten en cuenta las tareas que el jugador tiene que asumir. ¿Cuál es su objetivo? ¿Cómo alcanza la victoria? ¿Está disparando a cosas, procede a su recogida, las coloca en un lugar, previene que suceda algo? ¿Está moviendo cosas alrededor de la zona del juego con algún propósito? La respuesta a estas preguntas unidas con una técnica de control diferente es donde el juego puede empezar a diferenciarse de los innumerables juegos de plataformas, shoot-em-ups, juegos de laberinto o de puzle que hay desarrollados. Empieza de forma sencilla en un primer momento y ve añadiendo ideas a medida que avanzas. No tengas miedo de llegar a puntos donde ya no funciona, incluso si has pasado horas agonizando sobre tu código. Si una mecánica no funciona, no tiene cabida en tu juego.

El tercer paso es considerar los peligros que enfrentará el jugador. ¿Hay un tiempo límite? ¿Hay enemigos que se mueven por el juego? ¿Cómo se mueven, de manera predecible o de otra forma más interesante? ¿Son mortales al contacto o solo hacen que disminuya la energía del jugador? ¿Deben ser destruidos, deben evitarse o ambas cosas? ¿Cambian a medida que avanza el juego? ¿Se pueden utilizar de alguna manera?

Cuando hayas respondido a estas preguntas es necesario considerar si el jugador puede recoger pequeñas bonificaciones, o debe completar mini-tareas que le dan habilidades extra, aumentar su puntuación o su bonificación, aumentar vidas o ayudarlo de alguna otra manera. ¿Esto se debe lograr en un orden determinado para obtener una ventaja aún mayor? Puedes también explorar el camino contrario: ¿existen elementos que pueden ser recogidos que dificultan al jugador?, por ejemplo, con la inversión de sus controles o su fuerte desaceleración. Un error común (que he cometido yo mismo) es reducir las oportunidades de bonificación del jugador conforme el juego progresa. A medida que el juego se vuelve más difícil de jugar, generalmente es buena idea aumentar las bonificaciones con el fin de dar al jugador una oportunidad en el combate.

Si realmente quieres ir a la ciudad, es posible que desees añadir un poco más de profundidad. ¿Debe el jugador considerar una imagen más grande de la que ve en una sola pantalla? ¿Necesita pensar en algo mientras está esquivando, saltando o disparando? ¿Sus logros están llenando un mapa, o un cuadro? ¿Está jugando un juego gigante de tres en raya, de animal, vegetal o mineral, de batallas navales? ¿Puedes incluir elementos en el jardín mientras el jugador sigue su camino por su juego?

Juega mucho con tu juego mientras lo estás desarrollando y sigue haciéndote preguntas. Sé inseguro.

Cuando hayas hecho todo esto, es posible que desees considerar los gráficos, la historia o tal vez incluso la música. Evitar la tentación de comenzar con la historia cuando estás desarrollando un juego para una máquina de ocho bits que, simplemente, no tiene el manejo de gráficos o memoria suficientes para producir la atmósfera que haga justicia a una historia brillante. No vas a escribir el siguiente **Grim Fandango**. Adapta tu historia en torno al diseño del juego, y no al revés.

Por último, no importa lo bueno y original que sea tu juego, recuerda que no se puede contentar a siempre a todo el mundo. Simplemente disfruta de lo que estás haciendo, diviértete y no te lo tomes demasiado en serio.

¡Feliz desarrollo!

Apéndice A: Direcciones útiles de ROM y RAM

ROM

```
    0: inicio de la ROM.
  654: rutina ROM que devuelve la tecla pulsada (0-39) en el registro e,

       o 255 si nada presionado.
  949: rutina BEEPER.  Ajustar la duración en DE y el tono en el HL.
 3503: rutina para borrar la pantalla, estableciendo el color en (23693).
 6683: muestra en pantalla el valor numérico del par de registros BC,

       hasta un valor máximo de 9999.
 8252: muestra en pantalla una cadena de longitud BC ubicada en la dirección DE.
 8859: rutina para configurar el color del borde con el valor del acumulador.
15616: dirección de las fuentes en la ROM, 96 caracteres * 8 bytes.
```

RAM. Zona de pantalla

```
16384: 256x192 zona de pixeles.
22528:  32x24  zona de atributos de color.
```

RAM. Zona de variables

```
23296: variables del sistema.
23606: puntero a las fuentes, menos 256. (256 = 32 * 8 bytes, 32 es el código

       para el primer carácter imprimible)
23560: código ASCII de la última tecla pulsada.
23672: reloj, se incrementa 50 veces por segundo.
23693: PAPER/INK/BRIGHT color.
23695: PAPER/INK/BRIGHT color.
23734: canales de E/S.
23755: área del BASIC, seguido de espacio para programas en código máquina.

       Si usas hardware adicional puede moverlo.
24000: Podría decirse que es el punto de partida realista mas bajo para

       un juego, permitiendo 41536 bytes.
32767: último byte de RAM en un Spectrum de 16K.
32768: Principio de la zona no contenida, la RAM más rápida.
65535: último byte de RAM en un Spectrum de 48K.
```

APENDICE B: Edición y Ejecución de los Programas

Este apéndice no forma parte del documento original, es un añadido para explicar como se pueden probar los programas que se describen, y como ejecutar los que se incluyen para su descarga.

Para programar en ensamblador para el Spectrum en nuestro PC necesitamos un editor de textos, un programa ensamblador y un emulador. Esto se puede tener de forma separada o en un solo programa, lo que siempre es más cómodo, pero explicaré ambos sistemas.

Elegir un editor para programar

Lo único que hace falta para escribir código a la antigua usanza es un programa que permita escribir texto y un ensamblador. El **Notepad** incluido en windows es un editor muy sencillo pero suficiente, no hace falta más realmente, aunque hoy día estamos acostumbrados al coloreado del código o a que se auto complete (eso en ensamblador es menos útil), pero no tenemos porque renunciar a ello.

Si queremos un editor un poco mas potente existen muchas mas opciones, aunque no todas van bien con ensamblador del Z80 para el coloreado, pero como editores generales funcionan muy bien, existiendo muchos gratuitos muy buenos. Yo uso Notepad++ pero solo funciona en Windows (o en Linux con Vine), me parece muy bueno y completo, pero os doy otras opciones:

- SublimeText no es gratuito, es muy espartano de aspecto pero tiene muchos seguidores
- UltaEdit no es gratuito pero si muy completo, tiene una extensión llamada UltraCompare que va muy bien para comparar el contenido de ficheros.
- Atom está basado en Sublime pero es gratuito.
- Crimson Editor es gratuito y muy recomendado por programadores retro.
- Syncplify es gratuito, menos potente que los otros pero tiene un modo para su uso con pantallas táctiles, por lo que va muy bien en tablets.
- Textwrangler sería una buena alternativa si usamos Apple.
- Visual Studio Code de Microsoft es la alternativa mas profesional, y además es gratuito pero no lo recomiendo para principiantes, es un buen IDE pero tiene demasiadas cosas que no vamos a usar, dispone de varias extensiones para trabajar en ensamblador del Z80, incluso una para trabajar en el Basic del Spectrum.

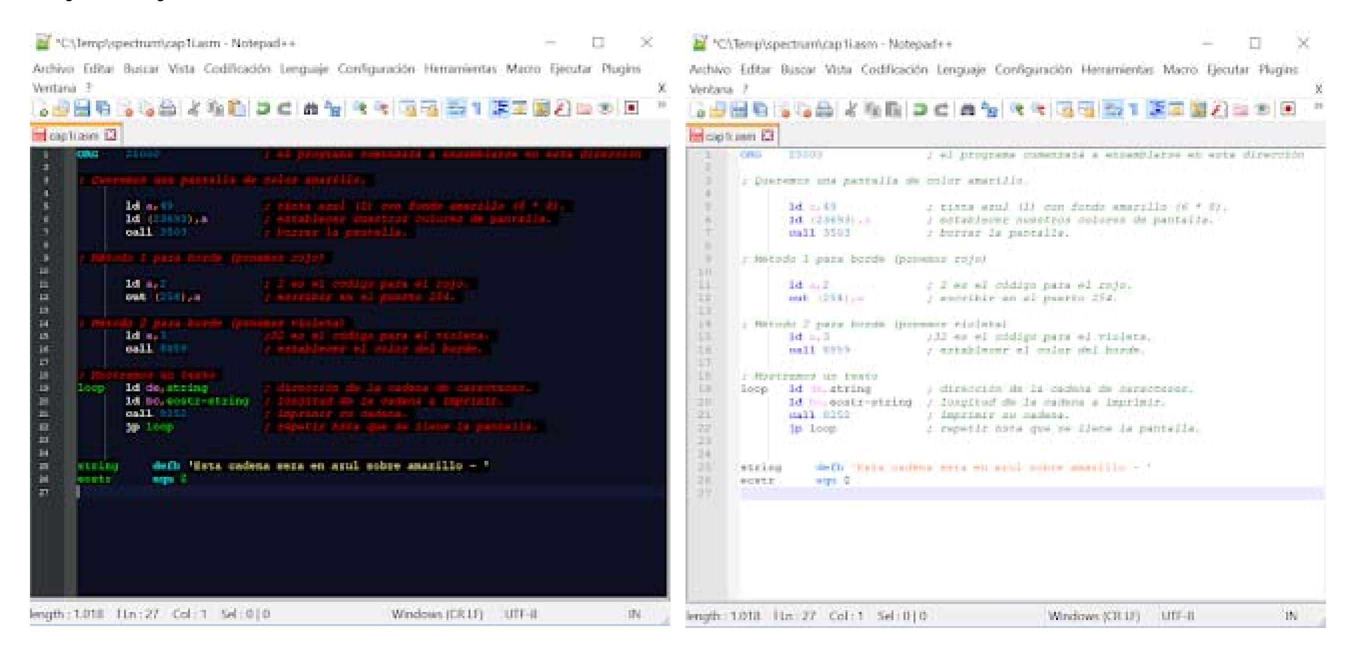
Configurar el Notepad++ para ensamblador del Z80

Notepad++ dispone de configuración específica para muchos lenguajes de programación, incluyendo ensamblador de Intel, pero como siempre se le pueden añadir mas, existen variantes para la sintaxis del Z80. Hay un fichero de configuración que podéis localizar en los foros de CPCWiki (está en inglés pero es muy sencillo de entender), que funciona muy bien. Está

desarrollado para usar fondo negro en la pantalla, yo he modificado el fichero para usar fondo blanco, podéis descarga de aquí los dos ficheros:

fondo negro   fondo blanco

y según mas os guste el fondo con el que trabajar ponéis uno u otro, incluso podéis trastear un poco y cambiar los colores a vuestro gusto, aquí tenéis el mismo ejemplo con ambos fondos para que elijáis mejor:



Para instalarlo, guardamos el que queramos (o los dos) en un directorio de nuestra maquina, abrimos Notepad++, vamos en el menú superior a "Lenguaje", opción "Defina su lenguaje...", en la pantalla nueva damos a importar, elegimos el fichero que queremos subir, y cerramos esa ventana. Ahora cerramos y abrimos Notepad++, y en la opción de "Lenguaje" nos aparecerá la opción importada, la seleccionamos y ya tenemos coloreado de sintaxis en el editor. También recomiendo quitar la vinificación gramatical, ya que cualquier directiva no la reconocerá y nos dirá que está mal escrita, para ello en la opción "Plugins", vamos a "DSpellCheck" y desmarcamos la opción "Spell Check Document Automatically".

Ahora podemos abrir un fichero existente o crear uno nuevo y comenzar a programar en ensamblador del Z80 con nuestro Notepad++

Generando el código máquina

Ahora que tenemos un programa escrito, hay que convertirlo en código máquina que pueda ejecutarse en el Spectrum, para lo que se usa un programa ensamblador específico para el Z80. Existen varios programas disponibles, el mas usado siempre ha sido PASMO, pero existe una buen alternativa llamada SjASMPlus, ambos están diseñado para Windows pero se pueden compilar para ejecutar en Linux o Apple ya que incluyen los fuentes, y como no usan ventanas sino que funcionan a partir de la línea de comando, son sencillos de portar. Los dos programas son antiguos y no se actualizan hace años, pero funcionan a la perfección.

Los ejemplos los he probado con ambos compiladores, la diferencia principal es que SjASMPlus admite mas directivas que PASMO, pero no permite que las directivas estén al principio de la línea por lo que hay que dejar espacios (se puede usar un parámetro al lanzarlo para oviarlo), y que es necesario añadir al inicio del programa una línea en la que se indica el nombre del fichero de destino como por ejemplo:

OUTPUT "destino.com"

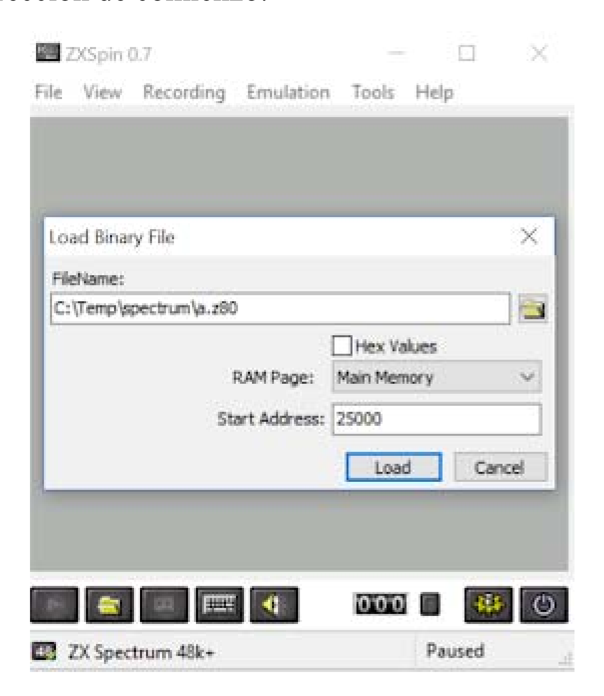
Ambos funcionan igual, hay que ir a la línea de comando de windows (usar buscar y escribir CMD), y lanzar la ejecución con estos parámetros, para PASMO origen y destino, para SJ solo origen:

- pasmo fuente.asm destino.com
- sjplusasm fuente.asm

Probando el programa en un emulador

Para ejecutar sirve cualquier emulador, Spectaculator tiene fama de ser el mejor, pero es de pago, yo prefiero usar ZXSpin (ya no tiene página propia, pero lo podéis descargar de aquí por ejemplo), ya que dispone de utilidades para desarrollo de programas en ensamblador, incluyendo editor, ensamblador y debuger, pero hay muchos donde elegir.

Descargas en programa, lo descomprimes y lo ejecutas, se abre una pantalla de Spectrum. En el menú de "File" seleccionas la opción "Load Binary File..." y se abre una pantalla, la rellenamos con el fichero generado y la dirección de comienzo:

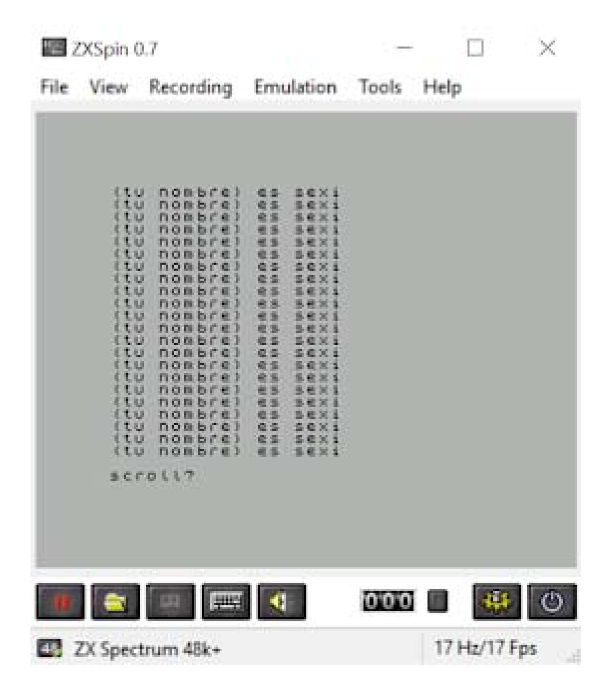


Luego hay que ejecutar el programa, por ejemplo para la dirección que hemos puesto debemos escribir: ***CLEAR 25000 : RANDOMICE USR 25000*** lo que se consigue en un spectrum con las

teclas X (CLEAR), el número deseado, Simbol Shift mas Z (:), T (RANDOMICE), Caps Shift mas Simbol Shift mas L (USR), y luego el número otra vez.



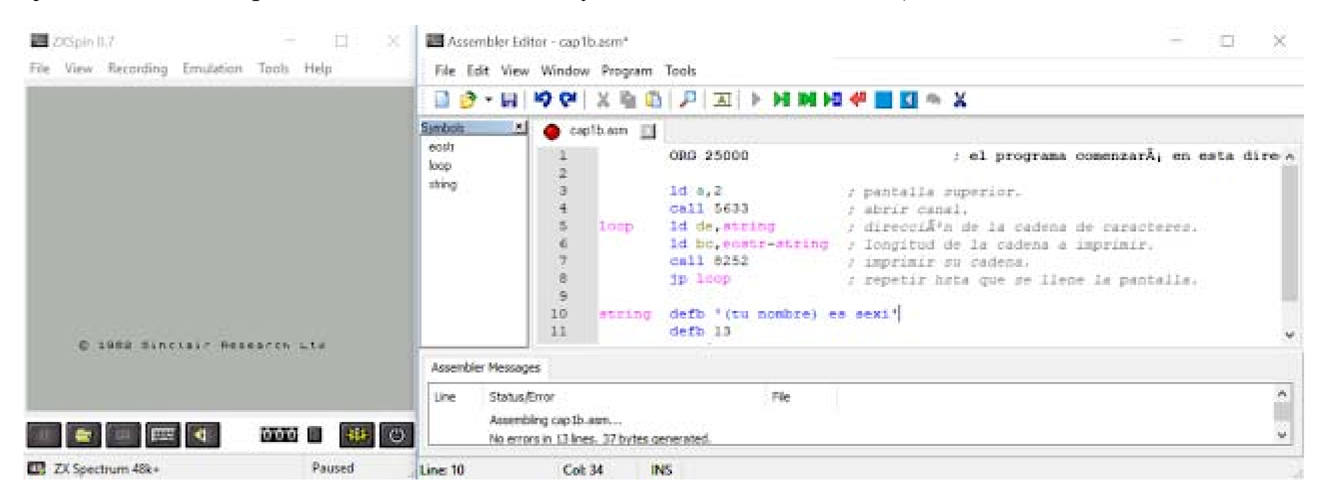
Esto ejecuta el programa cargado y podemos ver ya el resultado en la pantalla, este ejemplo es el hola mundo del primer capítulo:



Todo en uno con el emulador

si usamos ZXSpin o hace falta nada mas, ya que el emulador tiene su propio editor y ensamblador, para ello hay que ir a la opción "Tools", elegir "Z80 Assembler", y aparece una nueva ventana con el editor, podemos cargar allí el fuente, editarlo, ensamblarlo, ejecutarlo, incluso ejecutarlo paso a

paso, lo que nos ahorra un poco de trabajo. el editor no es la quinta maravilla, pero tiene coloreado y lista de símbolos navegable, podemos cargar o guardar programas en la opción File, ensamblarlos en la opción File (cuidado de poner en esa pantalla la dirección de comienzo para evitar errores de ejecución), y ejecutarlos en la opción Program (permite ejecutar normal, paso a paso, detener la ejecución en cualquier momento, o detener y reiniciar los contadores):



Un todo en uno, que da a veces problemas con el uso de la memoria del PC si a la hora de ensablar no le ponemos la dirección de comienzo, pues el programa no funciona todo lo fino que debería, pero una gran ayuda para el desarrollo y pruebas de los programas.